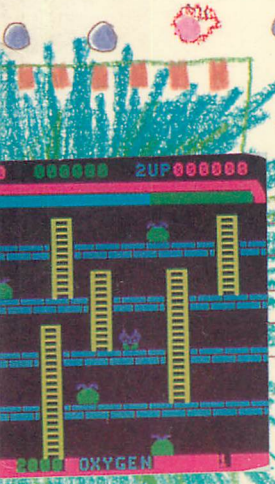
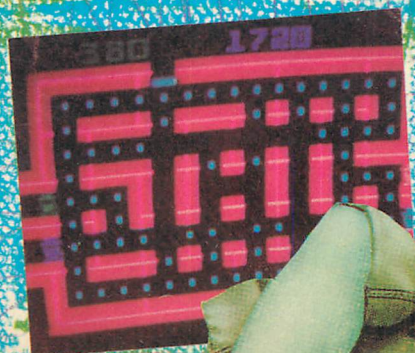
THE PENGUIN BOOK OF COMMODORE 64 GAMES

Robert Young and Paul Copeland





The Penguin Book of Commodore 64 Games

Robert Young is the author of a number of works on the Commodore 64 (including **How to Program the Commodore 64**, Interface Publications, 1983) and on other computers (**Dynamic Games for the Dragon 32**, Interface Publications, 1983). He became interested in computers after playing Adventure games against a university mainframe computer.

Mr Young, who lives in Geelong, Victoria, with his wife and son, is very interested in computer and non-computer simulation and role-playing games, and is currently working on a major book on historically accurate war-gaming. He left the public service in mid-1983 to concentrate fulltime on his writing career.

Paul Copeland, whose main area of interest in computers beyond that of game-playing is in the field of music composition, has received a Composition Grant from the Australian Council of the Arts, and has had music performed on the A.B.C. Paul started his musical studies at the Melba Conservatorium and in 1969 became a private student of Felix Werder. Paul's 'Continuum for Six' was performed in the Great Hall of Melbourne's National Gallery.

Series editor,

Melbourne-born Tim Hartnell is the most widely published author of computer books in the world. He runs his own specialist computer publishing house – Interface Publications – based in London, and contributes extensively to the computer press. He is founding editor of **ZX Computing** magazine. Books by Tim Hartnell include **The Giant Book of Computer Games** (Interface/Fontana, 1983; Ballantine, 1984); **Mastering the Timex Sinclair 1000** (Bantam Books, 1983); **Games BBC Computers Play** (Addison-Wesley/Interface, 1982) and **The Book of Listings** (BBC Publications, 1982). He now regards Australia as his home, and spends about seven and a half months of each year here, with the balance in America and Britain.

The Penguin Book of Commodore 64 Games

Robert Young
Paul Copeland

Edited by Tim Hartnell

Penguin Books

Penguin Books Australia Ltd,
487 Maroondah Highway, P.O. Box 257
Ringwood, Victoria, 3134, Australia
Penguin Books Ltd,
Harmondsworth, Middlesex, England
Penguin Books,
40 West 23rd Street, New York, N.Y. 10010, U.S.A.
Penguin Books Canada Ltd,
2801 John Street, Markham, Ontario, Canada
Penguin Books (N.Z.) Ltd,
182-190 Wairau Road, Auckland 10, New Zealand

First published by Penguin Books Australia, 1983

Copyright © Robert Young and Paul Copeland, 1983

Made and printed in Australia
by The Dominion Press-Hedges and Bell

All rights reserved. Except under the conditions described in the Copyright Act 1968 and subsequent amendments, no part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior permission of the copyright owner. Except in the United States of America, this book is sold subject to the condition that it shall not, by way of trade or otherwise, be lent, re-sold, hired out, or otherwise circulated without the publisher's prior consent in any form of binding or cover other than that in which it is published and without a similar condition including this condition being imposed on the subsequent purchaser.

CIP

Young, Robert.
The Penguin book of Commodore 64 games.
ISBN 0 14 007890 8.
1. Computer games. 2. Commodore computers –
Programming. I. Copeland, Paul. II. Title.
794.8'2

Contents

Introduction by Tim Hartnell vii

The Games

You-Draw 1
Noughts and Crosses 19
Evolution-Penultima 25
Space-Venture 31
Gamblers Delight 39
Oil Rig 47
Ten Pin Bowling 53
Runaround 57
Cubist 59
Magic Square 63
Cat and Mouse 65
Flip 69
Hangman 73
Breakout 77
Red Alert 81
Symbol Match 91
Al-Khwarizmi 101

The Appendices

Menu selection by cursor controls 109
Menu selection by function key 113
Timing input from the keyboard 117
Fast cursor positioning 121
Diet Helper 123
Runs per Over 127
Creating your own Games 131

I would like to thank Paul Copeland for many of the original programs and assistance he provided for this book. I also thank Kelvin Benson of Southern Office Equipment, Geelong. The support and back up he supplied was above and beyond the call of duty. Robert Young.

Introduction

This book was written with a sense of excitement, a sense of excitement which will transmit itself to you the moment you get the first program up and running. From SPACE VENTURE to RED ALERT, you'll find here a satisfying collection of challenging programs for your Commodore 64.

Robert Young and Paul Copeland have done their best to make the most of the color, sound and graphics potential of the computer, integrating the computer's capabilities into the programs.

Whether you're looking for board games which give solid evidence of your computer's intelligence, or a way to break the bank at Las Vegas, or you just want to put your machine to work to make sure it earns its keep, you'll find suitable programs here, just itching to get off the printed page and into your micro.

It's about time you caught the sense of excitement and gave way to the urging of the programs.

Time to turn the page, and start enjoying some great programs with your fun-loving Commodore 64.

Tim Hartnell,
Series editor,
Melbourne, 1983.

YOU-DRAW

This program allows the user to draw on a screen which is 22 characters wide and 23 characters high. The input commands are single key strokes. Any command will repeat by holding the key down. Once the drawing has been completed, it can be redrawn very quickly with different colors from the original drawing. A 'SAVE' command is available which will save the picture to tape for future retrieval.

Running the program.

Load the program from tape and type 'RUN' followed by RETURN. The screen will go black and a border will be drawn on the screen. You may now start drawing. As you draw, a number on the top left hand corner of the screen can be seen which is slowly reducing in value. This number indicates the number of keystrokes that remain.

Drawing mode.

Color control keys.

The keys numbered from 1-8 may be struck at any time to select the brush color written under the key. For example, key number 6 will select the color green. The color will always remain the same until a new color key is hit. The key numbered 9 is a special key which will activate a random color routine so that each brush movement is accompanied by a random brush color.

Brush positioning commands:-

"Q" ... positions brush at top left hand corner.
"Z" ... positions brush at bottom left hand corner.
"↑" ... positions brush at top right hand corner.
"/" ... positions brush at bottom right hand corner.
"H" ... positions brush at centre of screen.

Brush drawing commands (auto-repeat):-

"S" ... draw one character left (West).
"D" ... draw one character right (East).
"E" ... draw one character up (North).
"X" ... draw one character down (South).
"U" ... draw one character north-north-west.

"," ... draw one character south-south-east.
"N" ... draw one character south-south-west.
"O" ... draw one character north-north-east.

Save, load and redraw commands (can only be used whilst in the drawing mode).

"T" ... will send the picture to tape for storage.
"G" ... get a picture from the tape.
"P" ... redraw picture quickly and invoke "Re-draw" mode.

Re-draw mode.

"C" ... clears the screen to black and awaits further re-draw command.
"*" ... re-draws the picture with random colors.
"R" ... re-draws the picture with original colors.
"A" ... re-draws the picture with a different, but always similar set of colors.
"+" ... go back to draw mode. This will leave the current picture intact. The picture can be continued from where it was left off in the "drawing" mode.
"N" ... clears the screen and allows a new picture to be drawn.
"E" ... end the program.

Structure of the Program.

Numeric variables used:-

V%(707).
This array holds the picture screen POKE addresses together with the screen color information. The two types of information are stored as follows:- V%(X) holds screen POKE addresses for the brush position on the screen. V%(X+1) holds the color value for the previous screen (X) location.

Variable "A".
This variable changes in value and is used as a counter to fill the array V%(A) with screen information.

Variable "S".
This variable is set to equal 1, and is used to POKE the correct color to the screen. When POKEing colors to the screen it is necessary to subtract a 1 from the color number to arrive at the correct color. For example, to POKE the color green to a screen location, the number 5 should be poked.

Variable "T".
This variable is set to equal 40, and is used as the offset for north and south positioning of the brush.

Variable "E".
This variable is set to equal 86, which is the POKE code for the border character. This value is always PEEKed so that the brush will not draw past the border.

Variable "N".
This variable is set to equal 840. It is the offset which is added to or subtracted from the screen brush position when the screen border is about to be drawn on. This offset is used in north, south and diagonal brush moves.

Variable "B".
This variable set to 22. It is the offset which is added to or subtracted from the screen brush position when the screen border is about to be drawn on. This offset is used in east and west brush moves.

Variable "U".
This variable is set to 54272. This is the offset which is added to the screen character map to produce the color screen map. For example, to POKE a green color to the top left hand corner of the screen we would POKE 1024+U,5 where "U" equals the offset value of 54272.

Variable "H".
This variable is set to 91. This is the POKE code for the brush character.

Variable "J".
This variable is set to 2. It is used to increment the loop counter in the redraw mode and when a program is being saved.

Variable "K".
This variable is set to 1. It is used in various parts of the program, and like all of the other variables that remain the same, it is used to increase program speed.

Variable "X".
This variable represents the current screen character code for the brush position.

Variable "CX".
This variable always equals the value of the PEEk of memory location 203. This variable is used to inform the computer as to whether a key is currently being held down at any given moment.

String variables used:-

A$.
This variable is used to hold all of the input commands that are given by the user from the keyboard.

B$.
This variable is used at line 60. It is used to show whether a key has been held down.

Explanation of program listing.

Line #.

```
40 PRINT"■■";:POKE 53280,0:POKE 53281,0:GOSUB 890
```

Cursor color white. Clear screen. Screen color black. Border color black. Go to subroutine 890 and print "YOU-DRAW" in cyan.

```
50 DIMV%(999):R=1:S=1:T=40:E=86:N=840:B=22:U=54272:
   H=91:J=2:K=1
```

DIMension array and initialise variables.

```
60 X=1033:O=1:GOSUB80:X=1057:O=T:GOSUB80:X=1954:O=
   GOSUB80:X=1033:O=T:GOSUB80
```

The variable X is used here for the starting positions for the border drawing subroutine, the subroutine at line 80. The variable O is used as an offset to X when a vertical line is being drawn.

```
70 GOTO 90
```

The subroutine at line 80 is no longer required so the program jumps over it.

```
80 FOR I=1 TO 24:POKE X,E:POKE X+U,K:X=X+O:NEXT:
   RETURN
```

This is the border drawing subroutine which is called from line 60.

```
90 IF P=200 THEN 640
```

This variable is used as a flag to indicate whether a drawing has already been completed. The variable is set at line 480.

```
100 POKE 1445,H:POKE 1445+U,1
```

POKEs graphic character H (H=91) at the centre of the screen.

```
110 X=1445:C=2
```

The variable X holds the brush screen character position. Variable C holds the color code.

```
120 GET A$
```

Get a character from the keyboard. Start of direction command routine.

```
130 IF VAL(A$)=0 THEN 150
```

Checks the value of (A$). A value from 1-9 will be returned if any of the number keys have been entered. If the value is zero then jump to line 150. If value of A$ is larger than zero, then continue to next line number.

```
140 C=VAL(A$)
```

A number key has been struck (numbers 1-9) which represents a color code. The variable C will now hold the number value for later decoding in POKE statements to the color code memory.

Lines 150-190 are brush positioning commands.

```
150 IF A$="Q" THEN X=1115
```

If variable A$ holds a "Q" then variable X will be set to hold the value 1115 which is the screen position for the upper left hand corner.

```
160 IF A$="Z" THEN X=1875
```

If variable A$ holds a "Z" then variable X will be set to hold the value 1875 which is the screen position for the lower left hand corner.

```
170 IF A$="/" THEN X=1895
```

If variable A$ holds a "/" then variable X will be set to hold the value 1895 which is the screen position for the lower right hand corner.

```
180 IF A$="↑" THEN X=1135
```

If variable A$ holds a " ↑ " then variable X will be set to 1135 which is the screen position for the upper right hand corner.

```
190 IF A$="H" THEN X=1445
```

IF variable A$ holds a "H" then variable X will be set to 1445 which is the screen position for the centre of the screen.

Lines 200-470 are brush drawing commands.

```
200 IF A$<>"E" THEN 230
```

If the variable A$ is not equal to "E" then go to line 230.

```
210 IF PEEK(X-T)=E THEN X=X+N:GOTO 530
```

We drop through to this line only when A$ holds the character "E" which is a command to drawn one brush stroke north. We now PEEK at the proposed next brush position to see whether it contains the character 86 (variable E) which is a border character. If a border character is found then we add the offset N to X and reposition the brush at the bottom of the screen. We now go to line 530 where various housekeeping commands are carried out.

```
220 X=X-T:GOTO 530
```

We reach this line only if A$ is equal to "E" and the new proposed position does not contain a border character. The offset T is added to the screen position variable X. We now go to line 530.

```
230 IF A$<>"X" THEN 260
```

If variable A$ is not equal to "X" then go to line 260.

```
240 IF PEEK(X+T)=E THEN X=X-N:GOTO 530
```

Variable A$ holds "X". We now check if the new brush position (one position (one position south) contains a border character. If so, we add offset, and reposition cursor at top of screen.

```
250 X=X+T:GOTO 530
```

Variable A$ holds "X" and next position is not a border color. We reposition brush one character south.

```
260 IF A$<>"D" THEN 290
```

If A$ is not equal to "D" then go to 290.

```
270 IF PEEK(X+K)=E THEN X=X-B:GOTO 530
```

Variable A$ holds a "D". We now check next proposed square (one square east) for a border character. If a border character is found then reposition brush at leftmost position on screen. Now go to line 530.

```
280 X=X+K:GOTO 530
```

Variable A$ = "D" and next proposed position does not hold a border color. Draw one position right.

```
290 IF A$<>"S" THEN 320
```

If A$ is not equal to "S" then go to line 320.

```
300 IF PEEK(X-1)=E THEN X=X+B:GOTO 530
```

Variable A$ holds an "S". Check next proposed position (one character west) for border character. If border position exists then reposition brush at rightmost position. Now go to line 530.

```
310 X=X-1:GOTO 530
```

Variable A$ holds a "S" and next proposed position does not contain a border character. Move brush one character north-west. Now go to line 530.

```
320 IF A$<>"U" THEN 360
```

If variable A$ is not equal to "U" then go to line 360.

```
330 IF PEEK(X-T)=E THEN X=X+N:GOTO 530
```

Variable A$ holds a "U". Check next proposed position (north-west) for border character. If border character exists, then reposition brush furthest along south-east diagonal. Now go to line 530.

```
350 X=X-41:GOTO 530
```

Variable A$ contains a "U". Move brush one character north-west. Now go to line 530.

```
360 IF A$<>"," THEN 400
```

If variable A$ is not a "," then go to line 400.

```
370 IF PEEK(X+T)=E THEN X=X-N:GOTO 530
```

Variable A$ holds a ",". Check next proposed position (south-east) for border character. If border character exists then reposition brush furthest along north-west diagonal. Now go to line 530.

```
380 IF PEEK(X+K)=E THEN X=X-B:GOTO 530
```

Variable A$ holds a ",". Check next proposed position (east) for border character. If border character exists, then reposition brush furthest along north-west diagonal. Now go to line 530.

```
390 X=X+41:GOTO 530
```

Variable A$ holds a ",". Move brush one character south-east. Now go to line 530.

```
400 IF A$<>"O" THEN 440
```

If variable A$ is not a "O" then go to line 440.

```
410 IF PEEK(X-T)=E THEN X=X+N:GOTO 530
```

Variable A$ = "O". Check next proposed position (north-east) for border character. If border character exists then reposition brush north-east.

```
420 IF PEEK(X+K)=E THEN X=X-B:GOTO 530
```

Variable A$ holds a "O". Check next proposed position (east) for border character. If border character exists then reposition brush furthest along south-west diagonal. Now go to line 530.

```
430 X=X-39:GOTO 530
```

Variable A$ holds an "O". Move brush one character north-east. Now go to line 530.

```
440 IF A$<>"N" THEN 480
```

If variable A$ is not an "N" then go to line 480.

```
450 IF PEEK(X+T)=E THEN X=X-N:GOTO 530
```

Variable A$ holds "N". Check next proposed position (south-west) for border character. If border character exists then reposition brush furthest along north-east diagonal. Now go to line 530.

```
460 IF PEEK(X-1)=E THEN X=X+B:GOTO 530
```

Variable A$ holds a "N". Check next proposed position (west) for border character. If border character exists then reposition brush furthest along north-east diagonal. Now go to line 530.

```
470 X=X+39:GOTO 530
```

Variable A$ holds "N". Move brush one character south-west. Now go to line 530.

```
480 IF A$="P" THEN P=200:PRINT "□";:GOTO 60
```

If A$ holds a "P" then the drawing is completed. We set the flag variable P to equal 200. Now clear the screen and redraw the picture by going to line 60.

```
490 IF A$="T" THEN GOSUB 790
```

If A$ holds a "T" then go to subroutine starting at line 790 and save the current picture to tape.

```
500 IF A$="G" THEN GOSUB 840
```

If A$ holds a "G" then go to subroutine starting at line 840 and "GET" a picture from the tape.

```
510 GOTO 120
```

We arrive at this line only if none of the keys tested for have been struck. So we go back to line 120 and test for input again.

```
530 POKE X,H:POKE X+U,2
```

We arrive here if a direction command has been entered. So we POKE the screen position with screen character variable H (value=91). Now we POKE screen location plus offset "U" (value=54272) with black color (rubout).

```
540 PRINT "■       ":REM HOME SIX SPACES
```

Send cursor to home position and rubout number on top left of the screen. This number is decremented every time a direction command is given. It represents the number of cells in the array V%(X) remaining which can be used for drawing.

```
550 PRINT "■";INT(997-A)/2;
```

Send cursor home and print the number of cells remaining which can be used for drawing.

```
560 IF A=>997 THEN P=200:PRINT "□";:GOTO 60:
```

If the variable A is larger than or equal to 997 then we have run out of available cells for drawing, so we set the variable P flag to equal 200. Now we clear the screen and redraw the picture by going back to line 60.

```
570 IF C=1 AND A>3 THEN A=A-2
```

Check and see if the variable C is 1 and the variable A is larger than 3. If this is so then the brush color is black which means that the brush is moving but not drawing. We therefore decrement the counter variable A so that the array V%(X) is not filled with unnecessary brush moves.

```
580 IFC=9THENRC=INT(RND(1)*7+K):POKEX+U,RC:
    V%(A)=X:V%(A+K)=RC:A=A+2:GOTO 600
```

If the variable C is 9 then a random color is generated by letting variable RC equal a random integer between 1-7. The color map is POKEd with the color and the array V%(X) is updated with screen position and color number. The counter variable A is incremented by 2.

Lines 600-620 check for key being held down.

```
600 CX=PEEK(203):GET B$
```

The variable CX is used to hold the value in memory location 203. The GET B$ command clears the location of any garbage.

```
610 IF PEEK(203)=64 THEN GOTO 120
```

If the value in memory location is 64 then no key is being held down. We therefore go and get a key at line 120.

```
620 IF PEEK(203)=CX THEN GOTO 150
```

If the value in memory location 203 is the same as the value in variable CX (which holds the value from GET A$) then the same key is being held down. We therefore go and process the command by going to line 150.

```
630 GOTO 120
```

Go to drawing command input routine.

```
640 V%(A+K)=C
```

We will come here after the test in line 90 is made. The final color code (variable C) is put into array V%(X).

```
650 FOR I=1 TO A STEP J:POKE V%(I),H:POKE V%(I)+U,
    V%(I+K)-S:NEXT
```

This line redraws the picture. The variable A divided by 2 contains the total number of strokes that have been made to draw the picture. We POKE the screen with the contents of V%(I) which hold the screen position code. V%(I)+U represents the offset for the color map. V%(i+K)-S holds the color code.

Lines 660-730 check the keyboard for input of re-draw commands.

```
660 GET A$
```

Invoke keyboard input strobe.

```
670 IF A$="C" THEN PRINT "□":GOTO 660
```

If variable A$ holds a "C" then clear the screen to black and go to line 660 and await further re-draw commands.

```
680 IF A$="*" THEN GOSUB 760:GOTO 660
```

If variable A$ holds "*" then go to the random color drawing subroutine starting at line 760. Then go back to line 660 for another re-draw command.

```
690 IF A$="R" THEN S=0:GOTO 650
```

If variable A$ holds a "R" then change variable S to equal 0. The variable S contains the offset value. To draw with the proper colors, a 1 should always be subtracted from the color code variable C. By setting the variable S to 0 all colors will be one more in value than in the original drawing. For example, if the original color was purple (number 5), then the new color will be green (number 6). Now go to re-drawing routine starting at line 650.

```
700 IF A$="A" THEN A$="R":S=1:GOTO 650
```

If variable A$ is "A" then variable S is set to 1. This turns the color offset back to 'normal', so that the redrawn picture will be the same colors as in the original picture. Now go to re-drawing routine starting at line 650.

```
710 IF A$="+" THEN X=V%(A-2):GOTO 120
```

If variable A$ is a "+" then set the screen addresss variable X to the current screen address by subtracting a 2 from the array V%(X). Now go to line 120 and continue drawing.

```
720 IF A$="N" THEN CLR:RUN
```

IF variable A$ is a "N" then clear all variables and re-run the program.

```
730 IF A$="E" THEN PRINT "□□□□□□□□□" TAB(12)
    "■GOODBYE■ ■FOR■ ■NOW":END
```

If variable A$ holds a "E" then clear screen and print "GOODBYE FOR NOW" with reverse on and in CYAN. Turn the character color to white and end program.

```
740 GOTO 660
```

No legal command given so go back to line 660 and try again.

```
760 FOR I=1 TO A STEP J:POKE V%(I),H
```

This is the start of the random coloring routine which is invoked at line 680. The screen position V%(I) is POKEd with the screen character variable H (H=91).

```
770 IF PEEK(V%(I-1))<>PEEK(V%(I+K)) THEN R=INT
    (RND(1)*8)
```

We check here to see if the next drawing location is a different color, and if this is so we randomly choose a color which will be held in variable R.

```
780 POKE V%(I)+U,R:NEXT:RETURN
```

POKE color memory map with random color (variable R). When the for-next loop (starting at line 760) is completed, return to line 680.

Lines 790-830 contain routine for writing picture to tape for later retrieval.

```
790 GET A$:IF A$="" THEN 790
```

The computer waits here until a key is struck at the keyboard.

```
800 OPEN 1,1,1,A$:PRINT#1,A
```

Open tape as output device for writing to. The file name of the tape will be the single character that is held in variable A$. Send to the tape the value held in variable A which represents the number of cells used up in array V%(X) in drawing the picture.

```
810 FOR I=1 TO A:A$=STR$(V%(I)):PRINT#1,A$
```

We now send the numbers held in the array V%(X) to tape. The numbers are first changed to alphanumeric string variables (A$=STR$(V%(I))), for faster tape input.

```
830 NEXT:CLOSE 1:RETURN
```

When for-next loop starting at line 810 is finished, close the tape file and return to line 500.

Lines 840-880 contain the routine for reading a picture in from the tape.

```
840 GET A$:IF A$="" THEN 840
```

The computer waits until a key is struck. The input character should be the character name for the picture that is stored on tape.

```
850 OPEN 1,1,0,A$:INPUT#1,A
```

Open tape file named A$ for reading. Read variable A in increments of two. We get from tape two alphanumeric characters and store them in the variable A$ and B$.

```
860 FOR I=1 TO A-2 STEP J:INPUT#1,A$,B$
```

We now count from one to the value of A in increments of two. We get two alphanumeric characters from tape and store them in the variables A$ and B$.

```
870 V%(I)=VAL(A$):V%(I+K)=VAL(B$):POKE V%(I),H
```

The variable A$ is converted to a numeric variable and placed into array V%(X). The variable B% is converted to a numeric variable and placed into array V%(X). The screen is now POKEd with a character from the array V%(X).

```
880 POKE V%(I)+U,V%(I+K)-S:NEXT:CLOSE 1 RETURN
```

The screen color memory map is POKEd with a color. The for-next loop started at line 860 is incremented and when completed the tape file is CLOSEd and the program returned to line 840.

```
890 PRINT TAB(15) "■■YOU-DRAW■■";:RETURN
```

This subroutine prints the heading at the top of the screen and returns to line 50.

```
10 REM YOU DRAW
40 PRINT"■";:POKE 53280,0:POKE 53281,0:GOSUB 890
45 REM CLEAR SCREEN WHITE
50 DIMV%(707):A=1:S=1:T=40:E=86:N=840:B=22:U=54272:
   H=91:J=2:K=1
60 X=1033:O=1:GOSUB80:X=1057:O=T:GOSUB80:X=1954:O=1
   :GOSUB80:X=1033:O=T:GOSUB80
70 GOTO 90
80 FOR I=1 TO 24:POKE X,E:POKE X+U,K:X=X+O:NEXT:
   RETURN
90 IF P=200 THEN 640
100 POKE 1445,H:POKE 1445+U,1
110 X=1445:C=2
120 GET A$
130 IF VAL(A$)=0 THEN 150
140 C=VAL(A$)
150 IF A$="Q" THEN X=1115
160 IF A$="Z" THEN X=1875
170 IF A$="/" THEN X=1895
180 IF A$="↑" THEN X=1135
190 IF A$="H" THEN X=1445
200 IF A$<>"E" THEN 230
210 IF PEEK(X-T)=E THEN X=X+N:GOTO 530
220 X=X-T:GOTO 530
230 IF A$<>"X" THEN 260
240 IF PEEK(X+T)=E THEN X=X-N:GOTO 530
250 X=X+T:GOTO 530
260 IF A$<>"D" THEN 290
270 IF PEEK(X+K)=E THEN X=X-B:GOTO 530
280 X=X+K:GOTO 530
290 IF A$<>"S" THEN 320
300 IF PEEK(X-1)=E THEN X=X+B:GOTO 530
310 X=X-1:GOTO 530
320 IF A$<>"U" THEN 360
330 IF PEEK(X-T)=E THEN X=X+N:GOTO 530
340 IF PEEK(X-1)=E THEN X=X+B:GOTO 530
350 X=X-41:GOTO 530
360 IF A$<>"," THEN 400
370 IF PEEK(X+T)=E THEN X=X-N:GOTO 530
380 IF PEEK(X+K)=E THEN X=X-B:GOTO 530
390 X=X+41:GOTO 530
```

```
400 IF A$<>"O" THEN 440
410 IF PEEK(X-T)=E THEN X=X+N:GOTO 530
420 IF PEEK(X+K)=E THEN X=X-B:GOTO 530
430 X=X-39:GOTO 530
440 IF A$<>"N" THEN 480
450 IF PEEK(X+T)=E THEN X=X-N:GOTO 530
460 IF PEEK(X-1)=E THEN X=X+B:GOTO 530
470 X=X+39:GOTO 530
480 IF A$="P" THEN P=200:PRINT "□";:GOTO 60:REM
    CLEAR SCREEN
490 IF A$="T" THEN GOSUB 790
500 IF A$="G" THEN GOSUB 840
510 GOTO 120
520 REM END OF DRAW COMMANDS
530 POKE X,H:POKE X+U,2
540 PRINT "■      ":REM HOME SIX SPACES
550 PRINT "■";INT(997-A)/2;
560 IF A=>997 THEN P=200:PRINT "□";:GOTO 60:REM
    CLEAR SCREEN
570 IF C=1 AND A>3 THEN A=A-2
580 IFC=9THENRC=INT(RND(1)*7+K):POKEX+U,RC:V%(A)=X
    :V%(A+K)=RC:A=A+2:GOTO 600
590 POKE X+U,C-1:V%(A)=X:V%(A+K)=C:A=A+2
600 CX=PEEK(203):GET B$
610 IF PEEK(203)=64 THEN GOTO 120
620 IF PEEK(203)=CX THEN GOTO 150
630 GOTO 120
640 V%(A+K)=C
650 FOR I=1 TO A STEP J:POKE V%(I),H:POKE V%(I)+U,
    V%(I+K)-S:NEXT
660 GET A$
670 IF A$="C" THEN PRINT "□":GOTO 660:REM
    CLEAR SCREEN
680 IF A$="*" THEN GOSUB 760:GOTO 660
690 IF A$="R" THEN S=0:GOTO 650
700 IF A$="A" THEN A$="R":S=1:GOTO 650
710 IF A$="+" THEN X=V%(A-2):GOTO 120
720 IF A$="N" THEN CLR:RUN
730 IF A$="E" THEN PRINT "□■■■■■■■■■■" TAB(12) "
    ■GOODBYE■ ■FOR■ ■NOW":END
```

```
735 REM CLEAR SCREEN DOWN CURSOR REV ON CYAN
    REV OFFF
740 GOTO 660
750 REM REPLAY WITH RANDOM COLOURS
760 FOR I=1 TO A STEP J:POKE V%(I),H
770 IF PEEK(V%(I-1))<>PEEK(V%(I+K)) THEN R=INT
    (RND(1)*8)
780 POKE V%(I)+U,R:NEXT:RETURN
790 GET A$:IF A$="" THEN 790
800 OPEN 1,1,1,A$:PRINT#1,A
810 FOR I=1 TO A:A$=STR$(V%(I)):PRINT#1,A$
830 NEXT:CLOSE 1:RETURN
840 GET A$:IF A$="" THEN 840
850 OPEN 1,1,0,A$:INPUT#1,A
860 FOR I=1 TO A-2 STEP J:INPUT#1,A$,B$
870 V%(I)=VAL(A$):V%(I+K)=VAL(B$):POKE V%(I),H
880 POKE V%(I)+U,V%(I+K)-S:NEXT:CLOSE 1:RETURN
890 PRINT TAB(15) "■■YOU-DRAW■■";:RETURN
895 REM CYAN REV ON REV OFF WHITE
```

NOUGHTS AND CROSSES

This is the standard game of noughts and crosses in which a player takes turns playing against the computer.

The game is played on a board that the computer draws on the screen. Sophisticated keyboard checking techniques are used which will only allow the player to input legal moves. There is a help command available which will display the number values of the squares on the screen.

The computer plays defensively and will always block a winning move or alternatively choose the centre square if vacant or one of the corner squares. If none of these squares is available then the computer will choose from the remaining squares. A random move feature is provided to ensure that the games vary. When only one move remains, the computer will automatically make the move for the player.

Structure of the Program.

Lines 60-100. Draws playing board on the screen.
Lines 120-160. Players input. Only accepts legal move or help command.
Line 200. Subroutine to give help with screen.
Line 210. Subroutine to rubout help prompts.
Line 220-290. Updates screen with new position. Fills cell from array D(1-9) with either a 1 representing player's piece or a 5 representing the computer's piece.
Line 440. Compare possible winning positions with board position. If a winning position exists then go and tell player that he has won and ask if the player wishes to play another game.
Line 450. If a winning position exists then go and tell the player that the computer has won and ask if the player wishes to play another game.
Lines 470-510. The game is either won or drawn. Ask if the player wishes to play again.
Line 550. Check for possible winning move by the computer. If such a move exists then go to subroutine and do it.
Line 570. Check for winning move by player. If such a move exists then go to subroutine and block it.

Line 580. If the centre square is vacant then the computer will place an 'o' there.
Lines 59-69. These lines check for unique winning positions that are available for the player. If any position exists then the computer will block it.
Lines 700-860. If no win is imminent for the player, then the computer will randomly choose from one of the 4 corner squares.
Lines 870-900. If none of the corner squares is available then choose from the available remaining squares.
Lines 910-930. If these subroutines are called from line 550 then the computer plays the winning move. If these subroutines are called from line 570, then the computer blocks winning move.
Lines 940-970. Nested subroutines called from lines 710-860. The computer places a piece on one of the corner squares.

Suggestions for improvement:

: As there are over 300,000 combinations of moves available it is quite remarkable that the computer is very difficult if not impossible to beat. It may be advisable to have the computer play a beatable game by introducing more randomness into the move choices made by the computer.

: For very young children the game could be completely randomised thus allowing the computer to be beaten quite easily.

: As the computer moves quite quickly, a small music subroutine played whilst the computer makes its move would be entertaining.

```
10 REM NOUGHTS AND CROSSES
30 R$="■■■
```

```
60 PRINT "□" TAB(10) "■■NOUGHTS■■&■■CROSSES■■"
65 REM CLEAR SCREEN PURPLE REV ON RIGHT CRSR REV
   OFF WHITE
70 PRINT "■■■
```

```
270 PRINT "■D";:D(3)=5:RETURN
280 PRINT Y$+"■■■
```

```
560 NEXT:RESTORE
570 FOR Z=1 TO 8:READ A,B,C:IF FNA(X)=2 THEN 910
580 NEXT:IF D(5)=0 THEN D(5)=5:P$="5":RETURN
590 IF D(1)=1 AND D(9)=1 AND D(4)=0 THEN D(4)=5:
    P$="4":RETURN
600 IF D(4)=1 AND D(8)=1 AND D(7)=0 THEN D(7)=5:
    P$="7":RETURN
610 IF D(3)=1 AND D(8)=1 AND D(4)=1 AND D(1)=0 THEN
    D(1)=5:P$="1":RETURN
620 IF D(2)=1 AND D(4)=1 AND D(1)=0 THEN D(1)=5:
    P$="1":RETURN
630 IF D(2)=1 AND D(6)=1 AND D(3)=0 THEN D(3)=5:
    P$="3":RETURN
640 IF D(6)=1 AND D(8)=1 AND D(9)=0 THEN D(9)=5:
    P$="9":RETURN
650 IF D(1)=1 AND D(8)=1 AND D(4)=0 THEN D(4)=5:
    P$="4":RETURN
660 IF D(3)=1 AND D(4)=1 AND D(2)=0 THEN D(2)=5:
    P$="2":RETURN
670 IF D(3)=1 AND D(4)=1 AND D(8)=1 AND D(9)=0 THEN
    D(9)=5:P$="9":RETURN
680 IF D(3)=1 AND D(8)=1 AND D(6)=0 THEN D(6)=5:
    P$="6":RETURN
690 IF D(3)=1 AND D(7)=1 AND D(2)=0 THEN D(2)=5:
    P$="2":RETURN
700 R=INT(RND(1)*4+1):ON R GOTO 710,750,790,830
710 IF D(1)=0 THEN GOSUB 940:RETURN
720 IF D(3)=0 THEN GOSUB 950:RETURN
730 IF D(7)=0 THEN GOSUB 960:RETURN
740 IF D(9)=0 THEN GOSUB 970:RETURN
750 IF D(3)=0 THEN GOSUB 950:RETURN
760 IF D(7)=0 THEN GOSUB 960:RETURN
770 IF D(9)=0 THEN GOSUB 970:RETURN
780 IF D(1)=0 THEN GOSUB 940:RETURN
790 IF D(7)=0 THEN GOSUB 960:RETURN
800 IF D(9)=0 THEN GOSUB 970:RETURN
810 IF D(1)=0 THEN GOSUB 940:RETURN
820 IF D(3)=0 THEN GOSUB 950:RETURN
830 IF D(9)=0 THEN GOSUB 970:RETURN
```

```
840 IF D(1)=0 THEN GOSUB 940:RETURN
850 IF D(3)=0 THEN GOSUB 950:RETURN
860 IF D(7)=0 THEN GOSUB 960:RETURN
870 IF D(2)=0 THEN D(2)=5:P$="2":RETURN
880 IF D(4)=0 THEN D(4)=5:P$="4":RETURN
890 IF D(6)=0 THEN D(6)=5:P$="6":RETURN
900 IF D(8)=0 THEN D(8)=5:P$="8":RETURN
910 IF D(A)=0 THEN D(A)=5:P$=STR$(A):RETURN
920 IF D(B)=0 THEN D(B)=5:P$=STR$(B):RETURN
930 IF D(C)=0 THEN D(C)=5:P$=STR$(C):RETURN
940 D(1)=5:P$="1":RETURN
950 D(3)=5:P$="3":RETURN
960 D(7)=5:P$="7":RETURN
970 D(9)=5:P$="9":RETURN
```

EVOLUTION-PENULTIMA

This simulation is based on the game of LIFE by John Conway. It represents a colony of single cell life-forms and records their growth from generation to generation.

The cells live on a regimented grid and so each cell has a known number of neighbours. A cell in the centre of the colony can have up to eight neighbours, a cell in an outside row five and a corner cell can have three neighbours. The cells live by a strict code of behaviour which stipulates that:-

1. a cell with two or three neighbours will survive to the next generation.

2. a new cell will be born into an empty space which has three cells adjoining it.

3. any cell with four or more neighbours will die from overcrowding.

4. a cell with only one neighbour or a cell by itself will die of loneliness.

In this program the screen is divided up into four segements. Three are mirror images of the segment in the top left hand corner of the screen. This produces an excellent graphic display, which is enhanced by the color changes in the cells from one generation to the next.

Another interesting feature of Evolution-Penultima is that the starting colony of cells is created by the player and not randomly generated by the computer. To set up your colony, move the cursor, an open circle, around the screen using the cursor control keys. When you wish to place a cell into a location just press the RETURN key. After you have set up your colony of single celled animals, press the G key (for GO!) to start the group on its journey through life.

As you can see by looking at the listing the program has been written in modules, using plenty of REM statements to make it easy to understand.

```
10 REM       *****    EVOLUTION-PENULTIMA    *****
   (FOR COMMODORE-64)
20 :
30 REM              BY 1983 PAUL COPELAND
40 :
140 POKE 53280,0:POKE 53281,0:PRINT"□";
150 M=41:N=40:P=39:U=1:TT=22:FE=58:Z=2
160 O=54272:S=7:CH=87:CR=81:CO=1:U=1:C=86:T=354:
    G=403:H=32:L=54272
170 :
180 REM DEFINE FUNCTION FOR EVALUATING LIFE  AND
    DEATH
190 :
200 DEF FNA(X)=PEEK(O%(X)-M)+PEEK(O%(X)-N)+PEEK
    (O%(X)-P)
210 DEF FNB(X)=PEEK(O%(X)-U)+PEEK(O%(X)+U)+PEEK
    (O%(X)+P)+PEEK(O%(X)+N)
220 DEF FND(X)=PEEK(O%(X)+M)
230 DEF FNC(X)=FNA(X)+FNB(X)+FND(X)
240 :      CTRL 9 → CTRL 6 (GREEN)
250 PRINT TAB(8);"■ EVOLUTION--PENULTIMA ";:REM
    (RVS-ON GRN)
260 :
270 REM DIMENSION ARRAYS FOR HOLDING SCREEN
    INFORMATION, COLOR INFORMATION
280 :
290 DIM O%(180),R%(180),I%(180),RI%(180),H%(180),
    CO%(15)
300 :
310 REM FILL ARRAY CO%(1-15) WITH COLOR INFORMATION
320 :
330 FOR X=1 TO 15:CO%(X)=X:NEXT
340 :
350 REM DRAW BORDER
360 :
370 FOR X= 1064 TO 1102
380 POKE X,C:POKE X+O,S:NEXT
390 FOR X= 1944 TO 1982
```

RVS ON

```
400 POKE X, C:POKE X+0,S:NEXT
410 FOR X=1064 TO 1944 STEP 40
420 POKE X,C:POKE X+0,S:NEXT
430 FOR X=1102 TO 1983 STEP 40
440 POKE X,C:POKE X+0,S:NEXT
450 FOR X=1504 TO 1542
460 POKE X,C:POKE X+0,S:NEXT
470 FOR X=1083 TO 1964 STEP 40
480 POKE X,C:POKE X+0,S:NEXT
490 :
500 REM INITIALISE ARRAYS O%(1-180),R%(1-180),
    I%(1-180),RI%(1-180) WITH SCREEN
510 REM LOCATIONS
520 :
530 X=1105:F=U
540 FOR R= 1 TO 10:FOR C= 1 TO 18
550 O%(F)=X:X=X+U:F=F+U:NEXT:X=X+TT:NEXT
560 X=1141:F=U
570 FOR R= 1 TO 10:FOR C= 1 TO 18
580 R%(F)=X:X=X-U:F=F+U:NEXT:X=X+FE:NEXT
590 X=1905:F=U
600 FOR R= 1 TO 10:FOR C= 1 TO 18
610 I%(F)=X:X=X+U:F=F+U:NEXT:X=X-FE:NEXT
620 X=1941:F=U
630 FOR R= 1 TO 10:FOR C= 1 TO 18
640 RI%(F)=X:X=X-U:F=F+U:NEXT:X=X-TT:NEXT
650 :
660 REM FILL SCREEN COLOR MAP WITH COLOR
670 :
680 FOR X=1 TO 180
690 POKE O%(X)+0,CO:POKE R%(X)+0,CO:POKE I%(X)+0,
    CO:POKE RI%(X)+0,CO:NEXT
700 X=U
710 POKE O%(X),CH:POKE R%(X),CH:POKE I%(X),CH:POKE
    RI%(X),CH
720 :
730 REM GET INPUT FROM KEYBOARD
740 :
750 GET A$:IF A$="" THEN 750
760 :
770 REM KEY STRUCK SO ACT UPON COMMAND
```

```
780 :
790 IF PEEK(O%(X))<>CR THEN POKE O%(X),H:POKE
    R%(X),H:POKE I%(X),H:POKE RI%(X),H
800 :
810 REM MOVE RIGHT
820 :
830 IF A$<>"■" THEN 900: REM CSR-RIGHT
840 X=X+1: IF X>180 THEN X=1
850 IF PEEK(O%(X))<>32 THEN 840
860 POKE O%(X),CH:POKE R%(X),CH:POKE I%(X),CH:POKE
    RI%(X),CH
870 :
880 REM MOVE LEFT
890 :
900 IF A$<>"■" THEN 970: REM CSR-LEFT
910 X=X-1: IF X<1 THEN X=180
920 IF PEEK(O%(X))<>32 THEN 910
930 POKE O%(X),CH:POKE R%(X),CH:POKE I%(X),CH:POKE
    RI%(X),CH
940 :
950 REM MOVE UP
960 :
970 IF A$<>"■" THEN 1020: REM CSR-UP
980 X=X-18: IF X<1 THEN X=X+180
990 IF PEEK(O%(X))<>32 THEN 980
1000 POKE O%(X),CH:POKE R%(X),CH:POKE I%(X),CH:
     POKE RI%(X),CH
1010 :
1020 REM MOVE DOWN
1030 :
1040 IF A$<>"■" THEN 1090:REM CSR-DOWN
1050 X=X+18:IF X>180 THEN X=X-180
1060 IF PEEK(O%(X))<>32 THEN 1050
1070 POKE O%(X),CH:POKE R%(X),CH:POKE I%(X),CH:
     POKE RI%(X),CH
1080 :
1090 REM DEPOSIT A CELL
1100 :
1110 IF A$<> CHR$(13) THEN 1130: REM CARRIAGE
     RETURN
```

```
1120 POKE O%(X),CR:POKE R%(X),CR:POKE I%(X),CR:
     POKE RI%(X),CR
1130 IF A$="G" THEN 1180: REM GO AND START
     EVOLUTIONS
1140 GOTO 750
1150 :
1160 REM EVOLUTIONS
1170 :
1180 E=1
1190 PRINT "                                    ";
1200 PRINT"";TAB(11);"  EVOLUTION-NUMBER";E;
1210 :
1220 REM CALCULATE BIRTHS AND DEATHS
1230 :
1240 FOR X=1 TO 180
1250 IFFNC(X)=T ANDPEEK(O%(X))=CR OR FNC(X)=
     GANDPEEK(O%(X))=CRTHENH%(X)=U:GOTO1280
1260 IF FNC(X)=G AND PEEK(O%(X))=H THEN H%(X)=U:
     GOTO1280
1270 IF FNC(X)>G  OR FNC(X)<T THEN H%(X)=Y
1280 NEXT
1290 :
1300 REM UPDATE SCREEN
1310 :
1315 GOSUB 1500
1320 FOR X=1 TO 180
1330 IF H%(X)<>U THEN 1360
1340 POKE O%(X),CR:POKE O%(X)+L,CO%(Z):POKE R%(X),
     CR:POKE R%(X)+L,CO%(Z)
1350 POKEI%(X),CR:POKEI%(X)+L,CO%(Z):POKERI%(X),
     CR:POKERI%(X)+L,CO%(Z):GOTO1370
1360 POKE O%(X),H:POKE R%(X),H:POKE I%(X),H:POKE
     RI%(X),H
1370 NEXT
1380 Z=Z+U:IF Z=16 THEN Z=U
1390 :
1400 REM ZERO ARRAY H%(1-180)
1410 :
1420 FOR X=1TO180:H%(X)=V:NEXT
```

```
1430 :
1440 REM INCREMENT EVOLUTION NUMBER AND CREATE
     ANOTHER EVOLUTION
1450 :
1460 E=E+U:GOTO 1190
1470 END
1500 VV=54296:POKEVV,15
1505 FOR XV=15 TO 1 STEP -1
1507 POKEVV,XV
1520 W1=54276:W3=54290
1530 A1=54277:A3=54291
1540 S1=54278:S3=54292
1550 H1=54273:H3=54287
1560 L1=54272:L3=54286
1570 POKEA1,30:POKEA3,30
1580 POKES1,170:POKES3,170
1590 POKEW1,17:POKEW3,17
1600 POKEH1,15:POKEL1,107
1620 POKEH3,38:POKEL3,129
1625 FOR DD=1 TO 50:NEXT DD
1630 NEXT XV
1640 POKEW1,0:POKEW2,0:POKEW3,0
1650 POKEA1,0:POKEA2,0:POKEA3,0
1660 POKES1,0:POKES2,0:POKES3,0
1670 RETURN
```

SPACE-VENTURE

Welcome aboard. You are the commanding officer of the space shuttle "SS-VENTURE". Your space-ship is taking commodities to the eighteenth quadrant "SONTON", where you will be paid for the commodities that you deliver. During your journey, you will encounter the enemy who will try and thwart your mission. Their name is PLUTARK and they come from a distant galaxy ETHON.

The commodities that you are carrying are:-

MYON. This is a unit of time. When you start, you will have 43000 myons available for your journey. Because you are travelling very close to the speed of light, your myons are diminishing at the rate of 60 per second. If you deplete your store of myons before reaching the SONTON quadrant, you will have failed in your mission.

ORG. This is a unit of energy, which is used to power your space-ship. It is used at different rates depending on what activities you are involved in (direction controls shooting phasars and lasars). At the commencement of your journey you have 13000 orgs available. If you deplete your store of orgs before reaching the SONTON quadrant, you will have failed your mission.

SHIELD. Your shields are made of solar-nuclear plestonium which is a very rare metal. The shields are used to protect you against the damage caused when your enemy, the PLUTARKS, attack you. Whenever the ship is attacked an amount of shield is damaged. If your shields are depleted before reaching the 18th quadrant, you will have failed your mission.

VELK. Velk is a precious metal used by the plutarks in the production of their space-ships. As you travel through space you will be constantly firing lasars and phasars at the plutarks. Whenever your weapons hit the plutarks, you take on board pieces of velk. Velk is a highly valued treasure, and it is very important for you to get as much of it as possible.

Playing strategy.

When the game begins you will be in quadrant 1. The STATUS-MODULE will be displayed on the screen. You will have 9 lasers and 9

phasars to use. The display will show your x,y coordinate position and the direction of the plutark. By hitting any key, you will be taken to the COMMAND-MODULE. It is from this module you may issue an order. If you fire a laser or phasar and the plutark is within striking range your velk tally will increment by either 1 or 2. Your org supply will be depleted, and there is a possibility that the plutark may fire back and do some damage to your shields.

You may enter hyperspace at any time. This command will take you to another quadrant. You must be very careful with using this command, as a great deal of energy is used and there is always the possibility that you may be sent to an earlier quadrant.

Manouvering north, south, east and west takes up time and energy, so you should use these manouvers with care. If you should collide with a plu tark within the same quadrant, you will lose a fraction of your shield protection.

When moving forward to another quadrant, you must be careful that you are not too close to the plutark as he may fire upon you and do damage to your shields.

Should you succeed with your mission, you will be given a score which is based on the orgs, myons, and shields remaining, and the number of velks won. To set the highest score, you should aim at completing your mission in the shortest possible time, with a minimum of shield loss and a maximum number of velks on board.

Structure of the program.

Lines 20-80. Initialise variables and strings and define function.

Lines 90-200. Clear screen and give command-module prompts.

Line 210. Invoke GET$ command. Check if shields are zero or fuel is depleted or time has run out. If so then go to line 1090 and end program.

Line 220 Print variable TI (MYON).

Line 230 POKE white dot to screen and GOSUB 980 to play some music. POKE space to screen.

Lines 240-630. Routines for evaluating and acting on input from the keyboard (T N S E W L P F H).

Lines 67-720. Ship-status-module display.

Lines 730-850. Evaluation and display of enemy position.

Line 890. Invoke GET$ command. Any key hit will cause a return to the command-module.

Line 900. Update value of org. Check if shields are depleted, fuel depleted or time is exceeded. If so, go to line 1090 and display results.

Line 910. Display updated org value on screen.

Line 920. If no key has been struck, go back to line 890.

Line 930. If a key has been struck, go to command-module.

Line 970. Repositions player in different quadrant and changes grid position in the quadrant.

Line 980. Music subroutine.

Lines 1010-1025. Noise subroutine.

Lines 1030-1040. Noise subroutine.

Lines 1060-1075. Music subroutine.

Lines 1100-12309. Mission ended. Display results.

Lines 1100-1230. Mission ended. Display results.

```
10 REM SPACE VENTURE
20 POKE 53280,6:POKE 53281,2:PRINT "■":REM WHITE
25 VL=54296:WA=54276:AD=54277:SR=54278:HF=54273:
   LF=54272
30 F=15000:L=9:P=9:S=9:W=40000
40 DEFFNR(X)=INT(RND(1)*10+1)
```

```
50 A=FNR(X):B=FNR(X):C=FNR(X):X=FNR(X):Y=FNR(X)
   :Z=1:C=INT(RND(1)*3+Z)
60 X$="■■■
```

```
270 GOTO 660
280 IF A$<>"S" THEN 310
290 Y=Y+1:IF Y=11 THEN Y=1
300 GOTO 660
310 IF A$<>"E" THEN 340
320 X=X+1:IF X=11 THEN X=1
330 GOTO 660
340 IF A$<>"W" THEN 370
350 X=X-1:IF X=0 THEN X=10
360 GOTO 660
370 IF A$<>"L" THEN 450
380 L=L-1:IF L<0 THEN 210
390 GOSUB 1000
400 F=F-500
410 IF ABS(A-X)<3 AND ABS(B-Y)<3 AND ABS(C-Z)<3
    THEN H=H+1:GOSUB 960
420 IF RND(1)>.78 THEN S=S-.5:GOSUB 1030
430 IF S<1 THEN 1090
440 GOTO 660
450 IF A$<>"P" THEN 520
460 P=P-1:IF P<0 THEN 210
470 GOSUB 1030
480 IF ABS(A-X)<3 AND ABS(B-Y)<3 AND ABS(C-Z)<4
    THEN H=H+1:GOSUB 960
490 IF RND(1)>.73 THEN S=S-.7:GOSUB 1000
500 F=F-500
510 GOTO 660
520 IF A$<>"F" THEN 580
530 Z=Z+1:IF Z>17 THEN 1090
540 IFABS(A-X)<2ANDABS(B-Y)<2ANDABS(C-Z)<2THEN
    GOSUB1000:GOSUB1030:GOSUB1060
545 IFABS(A-X)<2ANDABS(B-Y)<2ANDABS(C-Z)<2THEN
    A=FNR(X):S=S-1
550 IF Z>C THEN C=INT(RND(1)*2+Z+1):A=FNR(X):
    B=FNR(X)
560 IF C=18 THEN C=17
570 GOTO 660
580 IF A$<>"H" THEN 650
590 IF Z=17 THEN Z=18:GOTO 1090
600 GOSUB 1060
610 F=F-750
```

```
620 Q=INT(RND(1)*2+1):Z=Z+Q:IF Z>18 THEN Z=17
630 S=S-1:IF Z>12 AND RND(1)>.4 THEN Z=Z-INT(Z/2)
640 GOTO 660
650 GOTO 210
660 PRINT "■";:REM CLEAR SCREEN
670 PRINT TAB(10) "■■SHIP■-■STATUS■-■MODULE":PRINT
675 REM YELLOW REV ON REV OFF
680 PRINT TAB(8) "■■MISSILES■  ■ORGS":PRINT:REM
    GREEN REV ON REV OFF
690 PRINT TAB(8) "■LASER■      ■SHIELDS":PRINT
700 PRINT TAB(8) "■PHASAR■     ■VELK■";:PRINT:PRINT:
    PRINT
705 REM REV ON REV OFF REV ON WHITE
710 PRINT TAB(10) "■■YOUR■ ■POSITION:-■":PRINT:REM
    YELLOW REV ON REV OFF
720 PRINT TAB(9) "■X=■■";X;"■■Y=■■";Y;"■■QUAD=■■";Z
725 REM REV ON WHITE REV OFF YELLOW REV ON REV OFF
    WHITE YELLOW REV ON REV OFF
727 REM WHITE
730 PRINT:PRINT:PRINT TAB(10) "■■ENEMY■ ■POSITION
    :-■■"
740 PRINT:IFA=XANDB=YANDC=ZTHENGOSUB980:GOSUB1030:
    GOSUB1060:GOSUB1060:GOTO760
750 GOTO 800
760 Z=1:A=FNR(X):B=FNR(X):C=INT(RND(1)*3+1)
770 F=F-500:S=S-1.5
780 S=S-1.5:IF S<1 THEN 1090
790 PRINT "■";:GOTO 670:REM HOME
800 PRINT TAB(6);:IF B>Y THEN PRINT "SOUTH ";
810 IF B<Y THEN PRINT "NORTH ";
820 IF A<X THEN PRINT "WEST ";
830 IF A>X THEN PRINT "EAST ";
840 IF C=Z THEN PRINT "OF YOU"
850 IF C>Z THEN PRINT "IN FRONT OF YOU"
860 IF C<Z THEN GOSUB 970
870 PRINT LEFT$(X$,16);LEFT$(Y$,4);L;SPC(8);S;
880 PRINT LEFT$(X$,16);LEFT$(Y$,6);P;SPC(8);H;
890 GET A$
900 G=F-INT(TI/10):IF S<1 OR G<1 OR TI>W THEN 1090
910 PRINT LEFT$(X$,24);LEFT$(Y$,2);G
```

```
920 IF A$="" THEN 890
930 IF A$<>"" THEN 90
940 GOTO 90
960 REM
970 Q=10-Z:C=INT(RND(1)*Q+Z):A=FNR(X):B=FNR(X):
    RETURN
980 POKEVL,15:POKEAD,17:POKESR,132:POKEWA,33
985 POKEHF,3:POKELF,134:FORDD=1TO250:NEXTDD:POKEHF,
    0:POKELF,0:FORDD=1TO250:NEXT
990 TD=TD+1:IFTD=3THENTD=0:RETURN
995 GOTO 980
1000 FORTY=1TO3:FORXX=15TO0STEP-1:POKEVL,XX:
     POKEWA,129:POKEAD,15:POKESR,136
1010 POKEHF,3:POKELF,35:FORDD=1TO25:NEXT:NEXT:NEXT
    :POKEWA,0:POKEAD,0:POKESR,0
1020 POKEHF,3:POKELF,35:FORDD=1TO25:NEXT:POKEWA,0
    :POKEAD,0:POKESR,0
1025 POKEHF,0:POKELF,0:RETURN
1030 POKEVL,15:POKEAD,17:POKESR,132:POKEWA,129:
     POKEHF,4:POKELF,73
1040 FORDD=1TO700:NEXTDD:POKEWA,0:POKEAD,0:
     POKESR,0:POKEHF,0:POKELF,0:RETURN
1060 POKEVL,15:POKEAD,17:POKESR,132:POKEWA,17:
     POKEHF,14:POKELF,24
1065 FORDD=1TO250:NEXT:GOSUB1400:POKEHF,18:
     POKELF,209
1070 FORDD=1TO250:NEXT:GOSUB1400:POKEHF,16:POKELF,
     195
1075 FORDD=1TO500:NEXT:POKEWA,0:POKEAD,0:POKESR,0:
     POKEHF,0:POKELF,0:RETURN
1090 REM END OF MISSION
1100 PRINT "□";:REM CLEAR SCREEN
1110 PRINT TAB(10) "■■MISSION■ ■OVER":MU=TI
1115 REM REV ON YELLOW REV OFF
1120 GOSUB 1060:GOSUB 1000:GOSUB 990:GOSUB 1060
1130 IF Z<18 THEN 1170
1140 PRINT:PRINT TAB(10) "■▙MISSION SUCCEDED!!":
     PRINT
```

```
1145 REM REV ON CYAN
1147 PRINTTAB(10) "■FUEL=■■";G:PRINT
1148 REM REV ON REV OFF WHITE
1150 PRINT "■MYONS=■■";W-MU:PRINT:PRINT"■▙
     SHIELDS=■■";S:PRINT:PRINT"■HITS=■■";H
1155 REM REV ON REV OFF WHITE REV ON CYAN
1160 PRINT:PRINT:PRINT"■■SCORE=■■";G+((MU-2000)*10)
     +(S*50)+(H*100):GOTO 1240
1165 REM YELLOW REV ON REV OFF WHITE
1170 PRINT:PRINT:IF MU<W THEN 1200
1180 PRINT TAB(10) "■MYONS■■ ■EXCEEDED!!!"
1185 REM REV ON REV OFF REV ON
1190 GOSUB 1230:GOTO 1240
1200 IF G>1 THEN 1220
1210 PRINT TAB(8) "■FUEL■■ ■DEPLETED!!":GOSUB 1230:
     GOTO 1240
1220 IF S<1 THEN PRINT "■SHIELDS■■ ■DEPLETED":GOSUB
     1230:GOTO 1240
1230 PRINT:PRINTTAB(8) "■▙VENTURE■■ ■FAILED!!■":
     RETURN
1235 REM REV ON CYAN REV OFF REV ON YELLOW
1240 GET A$:GOSUB 1060:IF A$="" THEN 1240
1250 END
1400 POKEHF,0:POKE LF,0:FORDD=1TO150:NEXT:RETURN
```

GAMBLERS DELIGHT

It is 1965 and you are in Las Vegas. The microchip has not yet been invented. This means that the one-armed bandit that you are feeding your hard earned dollars into is susceptible to being 'nudged' so that the wheels will stop spinning. You start with a bank of $100.00.

You may bet any amount at any time provided that you do not bet more than you have in the bank. Your gambling spree ends when you either run out of money or break the bank (upper limit $9999.00). If you get 3 of a kind the machine will pay you 3 to 1. If you get 4 of a kind the machine pays 25 to 1.

To 'nudge' the machine just hit any key while the wheels are spinning and a wheel will stop at either the position displayed or one turn further. The top left hand wheel is in need of repair, and occassionally it will stop if its own accord. As it is not easy to break the bank we suggest that you keep enough money for the trip back to your hotel. Good luck!!

Structure of the program.

Line 20. Define random number generator to produce any integer from 1 to the number included within the brackets.

Lines 30-60. Character strings holding wheel characters.

Lines 70-80. Cursor positioning character strings.

Line 100. Initilisation of variables.

Line 110. Fill array C%(1-4) with numbers which are used later to check for winning combinations.

Line 130. Clear screen and print title.

Lines 140-320. Draw one-armed bandit.

Lines 330-450. Input routine for player's bet. This routine checks the legality of the input (whether numerical 1-9, or less than/equal to bank).

Lines 480-510. Animation routine with sound for moving one-armed bandit's handle.

Line 560. Initialise the four counters P1, P2, P3, P4 with random numbers from 1-4. The variables hold the corresponding array graphic element from the string array C$(1-4).

Lines 590-670. Spin the wheels and get 'nudges' from the keyboard (GET A$).

Lines 690-730. Check for winning combinations and reward accordingly.

Line 740. If the bank (variable B) is 0 then go and tell player that he or she is broke and end game.

Line 750. If the bank (variable B) is larger than 9999 then go and tell the player that he or she has broken the bank and end game.

Lines 760-770. Game not over, so clear variables, update bank and spin the wheels again.

Lines 780-800. Flashing prompt subroutine which is called whilst the computer waits for the player to input bet.

Lines 820-890. Subroutines for updating the wheel positions on the screen.

Line 900. Subroutine to position the cursor at the Gth row and the Hth column.

Lines 910-920. Converts numeric variable B into alphanumeric variable A$ and strips it of the first character (space).

Lines 940-970. The game is over. The player has either won or lost, so we print the appropriate message. Wait for an input and play some music.

Line 980. Row and column data.

```
10 REM GAMBLERS DELIGHT
15 VL=54296:WA=54276:AD=54277:SR=54278:HF=54273:
   LF=54272
20 DEFFNR(X)=INT(RND(1)*X+1)
```

```
30 C$(1)="■◆◆■■■◆◆■":REM CYAN Z KEY DOWN LEFT CRSR
   WHITE
40 C$(2)="■◆◆■■■◆◆■":REM LT RED A KEY DOWN LEFT
   CRSR WHITE
50 C$(3)="■◆◆■■■◆◆■":REM YELLOW X KEY DOWN LEFT
   CRSR WHITE
60 C$(4)="■◆◆■■■◆◆■":REM ORANGE S KEY DOWN LEFT
   CRSR WHITE
70 X$="■■■
```

```
205 REM REV ON WHITE D I F KEYS REV ON YELLOW
210 PRINT TAB(11) "■  ■  ■     ■   ■  ■ ■"
215 REM REV ON WHITE REV OFF K REV ON YELLOW GREEN
    YELLOW
220 PRINT TAB(11) "■  ■  ■     ■   ■  ■  ■"
225 REM REV ON WHITE REV OFF K REV ON K YELLOW
    GREEN
230 PRINT TAB(11) "■  ■    ■■■■■■■   ■  ■  ■"
235 REM REV ON WHITE C REV OFF I*2 REV ON V YELLOW
    GREEN
240 FOR I=1 TO 2:PRINT TAB(11) "■  ■■■■■■■
```

```
380 IF V<=BANDASC(A$)=13THENG=15:H=20:GOSUB900:
    PRINT"■■BET=■■";Z$;:GOTO470
385 REM LT RED REV ON REV OFF WHITE
390 IF V<=B THEN 330
400 G=15:H=20:GOSUB 900:PRINT "■■CHEAT■  ";:Z$=""
410 FOR DD=1 TO 1000:NEXT
440 G=15:H=20:GOSUB 900:PRINT "       ";
450 G=15:H=20:GOSUB 900:PRINT "■■BET$";
460 GOTO 330
470 S=1771
480 FOR I=1 TO 7:GOSUB 1100
490 POKE S+0,5:POKE S,224:FOR D=1 TO 30:NEXT:
    POKE S,96:POKE S-1,96:S=S-40:NEXTI
500 FOR I=1 TO 7:GOSUB 1100
510 S=S+40:POKE S,224:POKE S-1,224:FOR D=1 TO 30:
    NEXT:POKE S,96:NEXT:POKE S,224
530 PRINT "■" TAB(10) "■■WHEELS--SPINNING":REM HOME
    LIGHT BLUE REV ON
540 B=B-V:IF B<0 THEN 940
550 GOSUB 910
560 P1=FNR(4):P2=FNR(4):P3=FNR(4):P4=FNR(4):R=FNR
    (T)+30
580 U=FNR(10)
590 GET A$:GOSUB 820:GOSUB 840:GOSUB 860:GOSUB 880:
    IF FNR(20)>19 THEN 620
600 IF A$="" THEN 590
610 IF U>4 THEN GOSUB 820
620 GET A$:GOSUB 840:GOSUB 860:GOSUB 880:IF A$=""
    THEN 620
630 IF U>5 THEN GOSUB 840
640 GET A$:GOSUB 860:GOSUB 880:IF A$="" THEN 640
650 IF U>6 THEN GOSUB 860
660 GET A$:GOSUB 880:IF A$="" THEN 660
670 IF U>7 THEN GOSUB 880
690 IF P1=P2 AND P2=P3 AND P3=P4 THEN B=B+(25*V):
    GOTO 740
700 IF P1=P2 AND P2=P3 THEN B=B+(3*V)
710 IF P1=P2 AND P2=P4 THEN B=B+(3*V)
```

```
720 IF P1=P3 AND P3=P4 THEN B=B+(3*V)
730 IF P2=P3 AND P3=P4 THEN B=B+(3*V)
735 FOR I=1 TO 1000:NEXT
740 IF B<1 THEN 940
750 IF B>9999 THEN B=9999:GOSUB 910:GOTO 940
755 GOSUB 1200
760 GOSUB 910:PRINT "■";TAB(10) G$:Z$="":REM HOME
770 V=0:G=18:H=20:GOSUB 900:PRINT "      ";:GOTO 330
780 IF X=2 THEN G=15:H=20:GOSUB 900:PRINT "     ";
790 FOR I=1 TO 20:NEXT:X=X+1:IF X=8 THEN X=0
800 IF X=7 THEN G=15:H=20:GOSUB 900:PRINT "■■BET$";
820 G=13:H=4:GOSUB 900:P1=P1+1:IF P1=5 THEN P1=1
830 PRINT C$(P1);:RETURN
840 G=18:H=4:GOSUB 900:P2=P2-1:IF P2=0 THEN P2=4
850 PRINT C$(P2);:RETURN
860 G=23:H=4:GOSUB 900:P3=P3+1:IF P3=5 THEN P3=1
870 PRINT C$(P3);:RETURN
880 G=18:H=9:GOSUB 900:P4=P4-1:IF P4=0 THEN P4=4
890 PRINT C$(P4);:RETURN
900 PRINT LEFT$(X$,G);LEFT$(Y$,H);:RETURN
910 A$=STR$(B):G=19:H=17:GOSUB 900:PRINT "     ■";:
    GOSUB 900
920 E=LEN(A$)-1:PRINT RIGHT$(A$,E);:RETURN
940 IF B<1 THEN PRINT "■" TAB(9) "■■■■YOU'RE-
    BROKE■■■■";:GOSUB 1200
945 REM HOME LT RED REV ON YELLOW LT RED
950 IF B>9998 THEN PRINT "■" TAB(10) "■■YOU-BROKE-
    THE-BANK";:GOSUB 1200
955 REM HOME REV ON ORANGE
960 GET A$:GOSUB 780:GOSUB 820:GOSUB 840:GOSUB 860:
    GOSUB 880:IF A$="" THEN 960
970 PRINT "■":CLR:RUN
980 DATA 13,4,18,4,23,4,18,9,19,17,15,20
990 END
1000 POKEVL,15:POKE AD,18:POKE SR,130
1010 POKE WA,17:POKE HF,15:POKE LF,205
1020 FOR DD=1 TO 500:NEXT
1030 POKE WA,0:POKE AD,0:POKE SR,0
1040 RETURN
1100 POKE VL,15:POKE AD,18:POKE SR,34:POKE WA,129:
     POKE HF,34:POKE LF,75
```

```
1110 FOR DD=1 TO 25:NEXT:POKE WA,0:POKE AD,0:
     POKE SR,0:RETURN
1200 POKE VL,15:POKE AD,17:POKE SR,132:POKE WA,33
1210 POKE HF,14:POKE LF,24:FOR DD=1 TO 300:NEXT:
     GOSUB 1500
1220 POKE HF,12:POKE LF,143:FOR DD=1 TO 250:NEXT:
     GOSUB 1500
1230 TD=TD+1:IF TD=3 THEN TD=0:POKE WA,0:POKE AD,0
     :POKE SR,0:RETURN
1240 GOTO 1210
1500 POKE HF,0:POKE LF,0:FOR DD=1 TO 150:NEXT:
     RETURN
```

OIL RIG

In this simulation you are the director of an oil mining company. Your objective is to find as many oil fields as is possible (maximum of 10) within the time limit. Once an oil field is found you can drill for oil. A depth indicator shows you the approximate depth to go. An oil field is worth $20,000.00. If you dig too deeply or too shallowly the value of the field will be lowered.

As you move around searching for oil you may make a test of the soil beneath the rig. This is done by striking the "/" key. Each test that you make costs the company $200.00 and you may not make more than 25 tests before finding each oil field.

You are charged $20.00 per move. In the oil business time is money, and the cost of running the business come to $20.00 per second. Whenever you make a test, there is the possibility that you may strike a mine. This would result in your rig being moved back to the starting position, costing time and money.

Each area searched has up to 20 mines in various places. When the simulation is over (either by running out of time or doing too many tests) you will be given a breakdown of the profit that your company has made. A profit of around $100,000.00 is quite good, and above $160,000.00 is excellent.

Structure of the Program.

Lines 30-40. Cursor positioning character strings.

Line 50. Depth indicator string and string of empty spaces used for 'rubbing out'.

Line 60. Define random number generator function. DIMension array R%(18,11). This array will hold the x,y coordinate positions for the oil wells and mines.

Line 70. Initialise variables.

Lines 80-200. Draw graphics on the screen. GOSUB 650 fills array R%(11,18) with 20 mines and 1 oil-field.

Line 210. Set time to "000000".

Line 220. Position cursor and print time.

Line 230. Position cusor in x,y matrix (top left hand corner) and draw a reversed space.

Line 240. Wait for key input and update time.

Line 250. If time is more than 6 minutes then go to next matrix.

Lines 260-300. Check for cursor positioning command and update screen accordingly. Add $20.00 to the cost (variable C).

Line 310. Check for test command (/). If so, go and test square.

Line 320. No legal key has been hit so we go back and try again.

Line 330. Cursor positioning routine.

Line 350-360. Clock updating subroutine. Includes a check for time elapsed.

Lines 370-450. A test command has been given at line 31. This subroutine increases the cost (variable C) by $200. It then checks to see if a mine is beneath the cursor, and if this is so the cursor is moved back to the starting position (top left hand corner of the matrix) with accompanying musical effects.

Lines 460-490. A test command has been given and the current position does not contain a mine. We now prompt the user with a direction number (1-4) indicating the approximate position of the oil field.

Lines 500-660. We arrive here if the cursor is above an oil-field (X=XC+1 AND Y=YC+5). We now prompt the user with a query for input for the depth of the drill. Once the depth has been typed in we update the profit variable (variable P) and increment the oil-rig number by one (variable O). If this variable equals 10 then the game is over and we go to the balance-sheet routine lines 670. Lines 610-620 clear the array R%(11,18) to zeros. Lines 650-660 now fill the array with 20 mines and 1 oil-field.

Lines 620-780. The game is over so we print the balance-sheet and wait for input. If a key is hit, the game can be played again.

```
10 REM OIL RIG
20 POKE 53280,4:POKE 53281,11:PRINT"■";:REM CLEAR
   SCREEN WHITE
30 X$="■■■
```

```
180 X=15:Y=0:GOSUB 330:PRINT O;
190 X=9:Y=4:GOSUB 330:PRINT T;
200 X=15:GOSUB 330:PRINT "$";:GOSUB 650
210 TI$="000000"
220 X=1:Y=4:GOSUB 350
230 X=2:Y=6:GOSUB 330:PRINT "■ ■";:REM REV ON REV
    OFF
240 GET A$:GOSUB 350:IF A$="" THEN 240
250 IF TI$>"000600" THEN 670
260 IF ASC(A$)=17 THEN GOSUB 330:PRINT ".";:Y=Y+1
    :C=C+20:IF Y=17 THEN Y=6
270 IF ASC(A$)=29 THEN GOSUB 330:PRINT ".";:X=X+1:
    C=C+20:IF X=20 THEN X=2
280 IF ASC(A$)=145 THEN GOSUB 330:PRINT ".";:Y=Y-1
    :C=C+20:IF Y=5 THEN Y=16
290 IF ASC(A$)=157 THEN GOSUB 330:PRINT ".";:X=X-1
    :C=C+20:IF X=1 THEN X=19
300 GOSUB 330:PRINT "■ ■";:X1=X:Y1=Y:X=15:Y=4:GOSUB
    330:PRINT C;:X=X1:Y=Y1
310 IF A$="/" THEN GOSUB 370
320 GOTO 240
330 PRINT LEFT$(X$,X);LEFT$(Y$,Y);:RETURN
350 PRINT LEFT$(X$,1);RIGHT$(Y$,4);MID$(TI$,3,2);
    RIGHT$(TI$,2);
355 IF TI$>"000600" THEN 670
360 RETURN
370 C=C+200:X1=X:Y1=Y:T=T+1:IF T>25 THEN 570
380 X=9:Y=4:GOSUB330:PRINT T;:X=X1:Y=Y1
390 IF R%((X)-1,(Y)-5)<>1 THEN C=C+250:GOTO 460
400 R%((X)-1,(Y)-5)=0:GOSUB330:PRINT"■■";:GOSUB900
    :GOSUB350:XA=X:YA=Y
405 REM LT RED 0 KEY WHITE
407 IF X=2 THEN 430
410 FOR X=XA TO 3 STEP -1:GOSUB 330:PRINT "■ ■";:
    GOSUB 350:REM REV ON REV OFF
420 GOSUB 330:PRINT ".";:NEXT
430 IF Y=6 THEN 460
```

```
440 FOR Y=YA TO 7 STEP-1:GOSUB330:PRINT"■ ■";:GOSUB
    350:GOSUB 330:PRINT ".";
445 REM REV ON REV OFF
450 NEXT:GOSUB 330:PRINT "■ ■";:RETURN
455 REM REV ON REV OFF
460 IFX>XC+1THENGOSUB330:PRINT"■▙2■■";:GOSUB900:
    GOSUB350:GOSUB330:PRINT "■ ■";
465 REM REV ON CYAN REV OFF WHITE
467 IF X>XC+1 THEN RETURN
470 IFX<XC+1THENGOSUB330:PRINT"■▙4■■";:GOSUB900:
    GOSUB350:GOSUB330:PRINT "■ ■";
475 REM REV ON CYAN REV OFF WHITE
477 IF X<XC+1 THEN RETURN
480 IFY>YC+5THENGOSUB330:PRINT"■▙3■■";:GOSUB900:
    GOSUB350:GOSUB330:PRINT "■ ■";
485 REM REV ON CYAN REV OFF WHITE
487 IF Y>YC+5 THEN RETURN
490 IFY<YC+5THENGOSUB330:PRINT"■▙1■■";:GOSUB900:
    GOSUB350:GOSUB330:PRINT "■ ■";
495 REM REV ON CYAN REV OFF WHITE
497 IF Y<YC+5 THEN RETURN
500 IF X=XC+1 AND Y=YC+5 THEN GOSUB 330:
    PRINT "■▙▒";
505 REM REV ON CYAN PLUS KEY USE COMM KEY
510 X1=X:Y1=Y:L=FNR(Z)+1:X=15:Y=18:GOSUB 330:PRINT
    "■■";RIGHT$(D$,L);
515 REM REV OFF YELLOW
520 X=9:Y=20:GOSUB 330:PRINT "■■DRILL■ ■■1■-■6■?";
525 REM REV ON GREEN REV OFF REV ON WHITE REV OFF
    REV ON REV OFF
530 GET A$:GOSUB350:X=19:GOSUB330:PRINT"?";:
    GOSUB330:PRINT" ";:IFA$="" THEN 530
540 IF VAL(A$)<1 OR VAL(A$)>6 THEN 530
550 D=ABS(VAL(A$)-Z)
560 P=P+20000-(D*2000)
570 O=O+1:T=0:X=9:Y=4:GOSUB330:PRINT"     ";:
    GOSUB330:PRINT T;:IFO=10 THEN 670
```

```
580 X=15:Y=0:GOSUB330:PRINT O;
590 X=15:Y=18:GOSUB330:PRINT RIGHT$(C$,6);
600 X=9:Y=20:GOSUB 330:PRINT RIGHT$(C$,11);
610 FOR X=1 TO 18:FOR Y=1 TO 11
620 R%(X,Y)=0:NEXT Y:GOSUB 350:NEXT X
630 GOSUB 650
640 X=X1:Y=Y1:GOSUB 330:PRINT".";:X=2:Y=6:GOSUB
    330:PRINT"■ ■";:REM REV ON OFF
650 FORI=1 TO 20:R%(FNR(18),FNR(11))=1:NEXT
660 XC=FNR(18):YC=FNR(11):R%(XC,YC)=5:Z=FNR(5)+1:
    RETURN
670 PRINT"■";SPC(12);"■■BALANCE■ ■SHEET■":PRINT
675 REM REV ON YELLOW REV OFF REV ON WHITE
680 PRINT:PRINT TAB(14) "■■EXPENSES":PRINT
685 PRINT:PRINT:PRINT
690 W=INT(TI/60)*20:TL=W+C
700 PRINT TAB(10) "■WAGES=$■■";W:PRINT
710 PRINT TAB(6) "■■TESTS/TRAVEL=$■■";C
720 TL$=STR$(TL)
730 PRINT:PRINT TAB(10) "■TOTAL=$■■ ";RIGHT$(TL$,
    LEN(TL$)-1):PRINT:PRINT
740 PRINT TAB(11) "■***■ ■PROFIT■ ■***":PRINT:PRINT
750 PRINT TAB(12) "$";P-TL:X=13:Y=18
760 GET A$:GOSUB 330:PRINT RIGHT$(C$,8);:GOSUB 330:
    PRINT "$";P-TL
770 IF A$="" THEN 760
780 CLR:RUN
900 FOR DD=1 TO 50:NEXT:RETURN
```

TEN PIN BOWLING

This program turns your Commodore 64 into a Ten Pin bowling alley. The pins are set up near the top of the screen and the game begins. You are shown a view of the alley from above.

You will be able to play a standard game of ten frames with two balls to the frame except in the cases where you have been good enough to knock down all the pins with your first ball. The computer will keep track of your score, tell you which frame and ball you are playing and reset the pins for you after each frame. The game also has a high score feature to add extra interest.

You will get a bonus score for knocking down all the pins with the two balls in the frame and a STRIKE if you knock down all the pins with the first ball.

After the computer has set up the pins for you (lines 770 to 850) it will then invite you to "PRESS RETURN TO BOWL". When you do this lines 860 to 930 will POKE a black ball onto the screen and make it trundle up to the pins accompanied by suitable sound effects. The rumble is provided by the subroutine from line 2000.

The game starts from line 40. Line 100 tells you that the computer is ready to bowl. The routine from 505 to 560 prints the results of the ball onto the screen.

```
10 REM TEN PIN BOWLING
15 REM "■" IS DOWN CURSOR
20 PRINT "■":REM CLEAR SCREEN
30 GOSUB 940
35 PRINT "■":REM CLEAR SCREEN
40 E=2
50 DIM A(10)
60 Y=0
65 S=0
70 FOR B=1 TO 10
80 GOSUB 770
90 FOR E=1 TO 2
95 IF E=1 THEN PRINT "■■■■■■■■■■■■■■■■■■FRAME:";B;
   "PRESS RETURN TO BOWL";
```

```
100 IF E=2 THEN PRINT "■■■■FRAME:";B;"PRESS RETURN
    TO BOWL";:REM CURSOR DOWN
400 INPUT A$
410 GOSUB 860
420 GOTO 505
440 Z=0
450 FOR C=1 TO 10
460 IF E=2 THEN GOTO 480
470 A(C)=79
480 IF RND(1)<.7 THEN A(C)=45
490 IF A(C)=45 THEN Z=Z+1
500 NEXT C
502 RETURN
505 PRINT "■":REM CLEAR SCREEN
506 GOSUB 440
510 PRINT TAB(16) CHR$(A(10));" ";CHR$(A(9));" ";
    CHR$(A(8));" ";CHR$(A(7))
520 FOR DD=1 TO 250:NEXT DD
530 PRINT TAB(17) CHR$(A(6));" ";CHR$(A(5));" ";
    CHR$(A(4))
535 FOR DD=1 TO 250:NEXT DD
540 PRINT TAB(18) CHR$(A(3));" ";CHR$(A(2))
550 FOR DD=1 TO 250:NEXT DD
560 PRINT TAB(19) CHR$(A(1))
565 PRINT "■■■■■■■■FRAME:";B;"BALL:";E
570 PRINT "■SCORE THIS FRAME:";Z
580 IF Z=10 THEN Z=15
590 IF Z=15 AND E=2 THEN PRINT "■*** BONUS ***"
595 IF Z=15 AND E=1 GOTO 720
600 IF E=2 THEN S=S+Z
610 IF E=2 AND B<>10 THEN PRINT "SCORE SO FAR IS:"
    ;S
620 FOR DD=1 TO 2000:NEXT DD
630 NEXT E
635 PRINT"■"
640 NEXT B
650 PRINT "SCORE SO FAR IS:";S
660 PRINT TAB(6) "■■■■■SCORE FOR THAT GAME WAS:";S
670 IF S<Y THEN 690
680 Y=S
```

```
690 PRINT TAB(8) "■■HIGHEST SCORE SO FAR:";Y
700 FOR DD=1 TO 3000:NEXT DD
710 GOTO 65
720 PRINT "■*** STRIKE ***"
730 FOR DD=1 TO 1000:NEXT DD
750 E=2
760 GOTO 600
770 FOR D=1 TO 10
780 A(D)=79
790 NEXT D
800 PRINT TAB(16) CHR$(A(10));" ";CHR$(A(9));" ";
    CHR$(A(8));" "CHR$(A(7))
810 PRINT TAB(17) CHR$(A(6));" ";CHR$(A(5));" ";
    CHR$(A(4))
820 PRINT TAB(18) CHR$(A(3));" ";CHR$(A(2))
830 PRINT TAB(19) CHR$(A(1))
840 FOR DD=1 TO 1000:NEXT DD
850 RETURN
860 FOR PP=1803 TO 1243 STEP -40
880 POKE PP,81:POKE PP+54272,0
890 FOR DD=1 TO 10:NEXT
900 POKE PP,32
910 GOSUB 2000
920 NEXT
930 RETURN
940 PRINT "■":REM CLEAR SCREEN
950 FOR X=0 TO 15:POKE 53280,X:FOR DD=1 TO 100
960 NEXT:NEXT
970 POKE 53281,8
980 FOR X=1 TO 5
985 PRINT "■"
990 PRINT TAB(8) "■■■■■■■■■■■■ TEN PIN BOWLING
    ■■■■":REM BLACK
1000 FOR DD=1 TO 500:NEXT:PRINT "■":FOR DD=1 TO
     250:NEXT DD
1010 NEXT X
1020 RETURN
2000 POKE 54296,15
2010 POKE 54278,17:POKE 54277,17
```

```
2020 POKE 54276,33
2030 POKE 54273,6:POKE 54272,108
2040 FOR DD=1 TO 10:NEXT
2050 POKE 54277,0:POKE 54278,0:POKE 54276,0
2060 RETURN
```

RUNAROUND

This short program is designed to give you a little mental exercise. The computer will display the numbers from zero to nine in a random order on the screen. Your task is to unscramble the numbers and put them into the correct order, 0 1 2 3 4 5 6 7 8 9, in as few moves as possible. This is done by either reversing the whole sequence or just a part of it. For example the computer has generated 7354098621. It will then ask you "WHICH NUMBER DO YOU WISH TO TRY". Entering 1 would then reverse the entire sequence to read 1268904537. Entering 5 would then reverse the numbers from the fifth number onwards, that is 1268735409. The game finishes when you have the numbers in sequence from 0 to 9.

The program performs this clever little juggling trick by placing the randomly chosen numbers (line 50) into a string (A$) this is done in lines 60 to 100. Line 70 checks to see that a number isn't picked twice.

The function CHR$ converts the number to its ASCII code so that it can be stored and printed out by the string A$. The program then uses MID$ and LEFT$ to manipulate the contents of the string after each input.

Line 130 prints the sequence of numbers onto the screen.

```
10 REM RUN AROUND
15 POKE 53280,11:POKE 53281,14
20 PRINT "■":REM CLEAR SCREEN
25 PRINT TAB(14) "■■■■■■■■■STAND BY":REM GRAY 1
   CURSOR DOWN
30 M=1:X=0:A$=""
40 FOR T=0 TO 9
50 L=INT(10*RND(1))+48
60 Q=1
70 IF MID$(A$,Q,1)=CHR$(L) THEN 50
80 IF Q<T THEN Q=Q+1:GOTO 70
90 A$=A$+CHR$(L)
100 NEXT
120 PRINT "■":PRINT TAB(8) "■■■■■■■■THIS IS MOVE
    NUMBER";M:REM BLUE
```

```
125 GOSUB 500
130 PRINT TAB(14) "■■■" A$:REM BLACK
140 PRINT TAB(4) "■■■WHICH NUMBER DO YOU WISH TO
    TRY";:REM RED
145 INPUT R:IF R>9 OR R<1 THEN 145
150 B$=""
160 FOR T=10 TO R STEP -1
170 B$=B$+MID$(A$,T,1)
180 NEXT T
190 A$=LEFT$(A$,R-1)+B$
200 IF A$="0123456789" THEN 220
210 M=M+1:GOTO 120
220 FOR DD=1 TO 1000:NEXT
225 PRINT "■"
230 PRINT TAB(9)"■■■■■■YOU DID IT IN";M;"MOVES"
240 POKE 53280,0:FOR DD=1 TO 250:NEXT
250 POKE 53280,6:FOR DD=1 TO 250:NEXT:GOTO 240
500 POKE 54296,15
510 POKE 54277,17:POKE 54278,17
520 POKE 54276,65
530 POKE 54275,1:POKE 54274,255
540 POKE 54273,19:POKE 54272,65
550 FOR DD=1 TO 250:NEXT
560 POKE 54276,0
570 POKE 54277,0:POKE 54278,0
580 RETURN
```

CUBIST

This program creates a two dimensional version of a well-known puzzle.

When the program runs, you will see a flat surface divided into 16 segments. Each of the segments will have a number from 1 to 4 on it. Thus each number appears four times.

The configuration which appears at the start of the game is the correct alignment of the segments. After a short delay the computer will then scramble the segments into a random order. Your task is to then twist the cube until it returns to its original state.

To twist a portion of the cube enter a number between 1 and 12. Here is the key to the moves available.

```
1     2     3     4
5     6     7     8
9    10    11    12
```

When you enter a number, the appropriate square, the one to its left, the one below it and the one below and to the left all rotate one position clockwise.

The initial arrangement of the cube is set up in lines 50 to 210. This is then shuffled by line 240.

The section 250 to 270 accepts your moves, checks the legallity of the twist you have requested and then goes on to lines 280 to 350 to perform the twist. The subroutine from line 400 prints out the cube on the screen.

```
10 REM CUBIST
12 PRINT "□":REM CLEAR SCREEN
15 DIM A(4),B(16)
17 POKE 53281,4:POKE 53280,11
20 F=0
30 B=1
40 FOR D=1 TO 4
```

```
50 C=0
60 IF B=1 THEN C=1
70 IF B=3 THEN C=2
80 IF B=9 THEN C=3
90 IF B=11 THEN C=4
100 B(B)=C
110 B(B+1)=C
120 B(B+4)=C
130 B(B+5)=C
140 C=0
150 IF B=1 THEN C=2
160 IF B=3 THEN C=6
170 IF B=9 THEN C=2
180 B=B+C
190 NEXT D
200 GOSUB 400
210 PRINT:PRINT TAB(11) "■■■STARTING POSITION":
    REM RED CURSOR DOWN
220 F=F+1
230 FOR T=1 TO 2000:NEXT T
240 IF F<11 THEN X=INT(16*RND(1))+1
250 IF F>10 THEN PRINT TAB(10) "■■ENTER YOUR MOVE";
    :REM BLUE
255 IF F>10 THEN INPUT X
260 IF X<1 OR X>16 THEN GOTO 240
270 IF X=5 OR X=9 OR X=1 OR X>12 THEN GOTO 240
280 A(1)=B(X)
290 A(2)=B(X+4)
300 A(3)=B(X+3)
310 A(4)=B(X-1)
320 B(X)=A(4)
330 B(X+4)=A(1)
340 B(X+3)=A(2)
350 B(X-1)=A(3)
360 GOSUB 400
370 IF F<10 THEN PRINT TAB(10) "■■ENTER YOUR MOVE";
    :REM BLUE
375 IF F<10 THEN INPUT U$
380 F=F+1
390 GOTO 240
400 PRINT "■":REM CLEAR SCREEN
```

```
410 PRINT:PRINT
415 GOSUB 500
420 FOR B=1 TO 16
430 PRINT TAB(12) "■" STR$(B(B));" ";:REM BLACK
440 IF B/4=INT(B/4) THEN PRINT
450 NEXT B
460 IF F<10 THEN PRINT TAB(10) "■■■THIS IS TWIST
    NO";F:REM YELLOW
470 IF F>10 THEN PRINT TAB(10) "■■■NUMBER OF MOVES'
    SO FAR";F-10:REM YELLOW
480 RETURN
500 POKE 54296,15
510 POKE 54277,17:POKE 54278,130
520 POKE 54276,17
530 FOR DD=1 TO 125:NEXT DD
540 POKE 54273,28:POKE 54272,224
550 POKE 54277,0:POKE 54276,0:POKE 54278,0
560 RETURN
```

MAGIC SQUARE

Magic Square is an exercise in mental arithmetic. The computer will generate a three by three grid. Some of the points on the grid will be occupied by numbers, others will be blank.

The computer challenges you to fill in the missing numbers. The only clue you need is that the totals of all the columns, both vertical and horizontal, are the same.

The numbers for the grid are set up by lines 40 to 260. Line 280 keeps a tally of the number of guesses you have had and lines 320 to 340 prints the grid on the screen.

```
10 REM MAGIC SQUARE
15 PRINT "□":REM CLEAR SCREEN
20 DIM A(9):DIM B(9)
30 U=-99:M=0
35 POKE 53280,11:POKE 53281,4
40 A=INT(9*RND(1))+1
50 J=0
60 B=INT(9*RND(1))+1
70 C=INT(9*RND(1))+1
80 IF A=B OR A=C OR B=C THEN GOTO 60
90 A(1)=A+B
100 A(2)=A-(B+C)
110 A(3)=A+C
120 A(4)=A-B+C
130 A(5)=A
140 A(6)=A+B-C
150 A(7)=A-C
160 A(8)=A+B+C
170 A(9)=A-B
180 FOR Z=1 TO 9
190 B(Z)=A(Z)
200 NEXT Z
210 K=ABS(A)
220 B(K)=0
230 K=ABS(B)
240 B(K)=0
```

```
250 K=ABS(C)
260 B(K)=0
270 J=J+1
275 PRINT "■":REM CLEAR SCREEN
280 PRINT TAB(9)"■■■■THIS IS GUESS NUMBER";J:REM
    DOWN CURSOR RED
310 FOR Z=1 TO 9
320 PRINT TAB(8) "■■■"  B(1),B(2),B(3):REM BLUE
330 PRINT TAB(8) "■"B(4),B(5),B(6)
340 PRINT TAB(8) "■"B(7),B(8),B(9)
350 IF M=9 THEN PRINT TAB(6) "■■■YOU HAVE SOLVED
    THE PROBLEM":REM RED
355 IF M=9 THEN PRINT TAB(14) "■IN";J;"GUESSES"
360 IF M=9 THEN GOTO 480
370 PRINT TAB(9) "■■■■YOU HAVE";M;"NUMBERS RIGHT"
    :REM GRAY 1
390 PRINT TAB(9) "■■■THE CORRECT ANSWER IS";:
    INPUT U:REM YELLOW
400 M=0
410 FOR Z=1 TO 9
420 IF U=-99 THEN GOTO 440
430 IF A(Z)=U THEN B(Z)=U
440 IF B(Z)<>0 THEN M=M+1
450 IF A(Z)=0 AND U=0 THEN M=M+1
460 NEXT Z
470 GOTO 270
480 FOR DD=1 TO 1500:NEXT
490 GOTO 30
```

CAT AND MOUSE

This is your big opportunity to be an ecologist. The game Cat and Mouse requires you to populate a desert island with cats and mice in such a way that the ecosystem becomes self supporting. If you have too many cats the mice won't be able to multiply fast enough to feed them. Too many mice and they will overrun the island and crowd out the cats.

Each month the computer will give you an updated report on your island. You will be told how many cats and mice still survive and how many months the experiment has been running.

Each time the game is RUN the calculation in line 100 sets the ratio of mice it takes to feed each cat. Because it is in the form of a random number generator the result will never be predictable.

Lines 300 to 320 work out how fast the cats and mice have breed and been eaten. The print lines from 330 to 340 tell you how well your menagerie is doing.

The variable HS in line 390 provides a HISCORE feature so you can try to improve your skills in animal husbandry.

```
 10 REM CAT AND MOUSE
 20 PRINT "■":REM CLEAR SCREEN
 22 POKE 53280,4:POKE 53281,11
 25 PRINT "■■■■■■■■■■■■■■■WELCOME TO CAT AND MOUSE"
    :REM CURSOR DOWN RIGHT WHITE
 27 FOR DD=1 TO 1500:NEXT:PRINT "■"
 29 POKE 53280,6
 30 PRINT "■■■IN THIS GAME YOU ARE A GREAT
    ECOLOGIST":REMCURSORDOWNRIGHT LT RED
 40 PRINT "■■IT IS YOUR JOB TO SET UP A SOCIETY IN"
 50 PRINT "■WHICH A POPULATION OF CATS AND MICE WILL"
 60 PRINT TAB(12) "WILL REMAIN STABLE"
 65 PRINT TAB(10) "■■■PRESS ANY KEY TO PLAY"
 70 GET A$:IF A$="" THEN GOTO 70
 80 HI=0
```

```
100 FD=RND(1)
105 POKE 53280,0:POKE 53281,7
110 PRINT TAB(6) "■■■■■■■HOW MANY CATS DO YOU WANT
    TO START WITH":REM RED
115 PRINT TAB(12) "■(LESS THAN 100)";
120 INPUT CP:IF CP>99 THEN 110
130 PRINT TAB(12) "■■■CAT POPULATION: ";CP
140 CP=CP/3
150 FOR DD=1 TO 1000:NEXT
200 PRINT "■":REM CLEAR SCREEN
205 POKE 53280,11:POKE 53281,14
210 PRINT TAB(4) "■■■■■■■NOW, HOW MANY MICE WILL
    WE HAVE":REM BLUE
215 PRINT TAB(12) "■(LESS THAN 100)";
220 INPUT MP:IF MP>99 THEN 210
225 PRINT TAB(12) "■■■MOUSE POPULATION: ";MP
227 FOR DD=1 TO 1000:NEXT
230 MP=MP/3
240 PRINT "■":REM CLEAR SCREEN
245 POKE 53280,6:POKE 53281,8
250 PRINT TAB(5) "■■■■■■■■■■I AM GETTING YOUR CATS
    AND MICE":REM BLACK
260 GOSUB 1000:PRINT "■"
270 DA=0:CS=CS/2:MS=MS/2
280 DA=DA+1
285 PRINT "■":GOSUB 600
287 POKE 53280,8:POKE 53281,6
290 PRINT TAB(14) "■■■■■■MONTH: ";DA:REM ORANGE
300 IF CP>MP/FD THEN CP=MP/FD
310 CP=CP+((8*CP-CP*MP/3)*FD)
320 MP=MP+((4*MP-MP*CP)*.01)
330 PRINT TAB(12) "■■■■■■■YOU HAVE";INT(CP);"CATS"
340 PRINT TAB(12) "■■■YOU HAVE";INT(MP);"MICE"
350 GOSUB 1000
360 IF MP<2 OR CP<2 THEN 390
370 GOTO 280
390 IF DA>HS THEN HS=DA
395 IF CP<2 OR MP<2 THEN PRINT "■"
397 POKE 53280,6:POKE 53281,10
400 PRINT TAB(12) "■■■■■■■END OF THE LINE":REM
    GRAY 1
```

```
410 PRINT TAB(3) "■THE SOCIETY SURVIVED FOR";DA;
    "MONTHS"
420 PRINT TAB(6) "■■YOUR BEST EFFORT SO FAR IS";HS
430 PRINT TAB(2) "■■INPUT Y FOR A NEW ISLAND OR ANY
    OTHER"
440 PRINT TAB(15) "■KEY TO STOP"
450 GET A$:IF A$="" THEN 450
455 IF A$="Y" THEN 105
460 PRINT "■":POKE 53280,0:POKE 53281,0
470 PRINT TAB(16) "■■■■■■■■■GOODBYE":REM YELLOW
480 GOTO 480
600 POKE 54296,15
610 POKE 54277,18:POKE 54276,17
620 FOR DD=1 TO 500:NEXT
630 POKE 54273,57:POKE 54272,172
635 POKE 54276,17
640 POKE 54277,0:POKE 54276,0
650 POKE 55276,0:POKE 54277,0
660 RETURN
1000 FOR DD=1 TO 3000:NEXT
1010 RETURN
```

FLIP

Flip is a simple little puzzle which can cause a great deal of thought and frustration. The computer places nine playing pieces on a three by three grid. There are two types of playing pieces, solid circles and open circles or rings. At the start of the game the computer randomly selects which type of piece is placed in a grid location. The solid circles can be turned over, or flipped, to show a ring, but it isn't possible to flip an open circle.

The object of the game is to end up with an open ring in the centre position of the grid and solid circles in the other eight positions.

Flipping circles isn't quite as simple as it might seem. To make the puzzle a little more interesting, when you flip a corner piece the computer also turns over the three adjoining pieces. That is, if you turn over piece number one, numbers 2, 4 and 6 also turn over whether they are solid circles or not. When you flip a piece in the centre of an outside row the pieces either side of it also turn over. When the centre piece is turned over, all the pieces in the centre of outside rows, the even numbered pieces, turn over with it.

The computer will display the grid on the screen with a key to the numbering system beside it. You will then be asked which piece you wish to flip. The computer will make the necessary adjustments to the playing pieces and tell you how many turns you have had.

Variables Q and X in line 10 set the graphic symbols for the game. By altering these values you can change the playing pieces to any shape you like.

The routine from line 20 to 40 pick the starting pieces.

Lines 310 to 470 hold the calculations which make the flips. Line 100 checks to see if you have solved the puzzle after each turn.

```
5 REM FLIP
7 PRINT"J":REM CLEAR SCREEN
10 M=0:Q=209:X=215
15 POKE 53281,11:POKE 53280,6
```

```
20 FOR C=1 TO 9
30 B=INT(RND(1)+.5)
35 A(C)=Q:IF B=0 THEN A(C)=X
40 NEXT C
50 GOSUB 180
60 N=0
70 FOR C=1 TO 9
80 IF A(C)=X THEN N=N+1
90 NEXT C
100 IF N=1 AND A(5)=X THEN GOTO 270
110 M=M+1
115 PRINT:PRINT:PRINTTAB(8)"■MOVE NUMBER"M:PRINT:
    PRINT:REM LT RED
120 PRINTTAB(8)"WHICH ONE TO FLIP"
125 GET A$:IF A$="" THEN 125
126 N=VAL(A$):IF N<1 OR N>9 THEN 125
130 GOSUB 310
140 GOTO 50
170 END
180 PRINT"■"
190 PRINT:PRINT:PRINT
200 PRINT TAB(8)"■"1" "2" "3" "CHR$(A(1))"
   "CHR$(A(2))"  "CHR$(A(3))
210 PRINT
220 PRINT TAB(8) "■"4" "5" "6" "CHR$(A(4))"
   "CHR$(A(5))"  "CHR$(A(6))
230 PRINT
240 PRINT TAB(8) "■"7" "8" "9" "CHR$(A(7))"
   "CHR$(A(8))"  "CHR$(A(9))
250 PRINT
260 RETURN
270 PRINT TAB(8) "■ SOLVED IT IN"M;"MOVES":REM
    YELLOW
280 FOR T=1 TO 4000:NEXT
300 RUN
310 IF A(N)=X THEN RETURN
320 IF N=1 THEN F(1)=2:F(2)=4:F(3)=5:F(4)=10
330 IF N=2 THEN F(1)=1:F(2)=3:F(3)=10:F(4)=10
```

```
340 IF N=3 THEN F(1)=2:F(2)=5:F(3)=6:F(4)=10
350 IF N=4 THEN F(1)=1:F(2)=7:F(3)=10:F(4)=10
360 IF N=5 THEN F(1)=2:F(2)=4:F(3)=8:F(4)=6
370 IF N=6 THEN F(1)=3:F(2)=9:F(3)=10:F(4)=10
380 IF N=7 THEN F(1)=4:F(2)=5:F(3)=8:F(4)=10
390 IF N=8 THEN F(1)=7:F(2)=9:F(3)=10:F(4)=10
400 IF N=9 THEN F(1)=8:F(2)=5:F(3)=6:F(4)=10
410 GOSUB 500
420 FOR G=1 TO 4
430 IF A(F(G))=X THEN A(F(G))=Q:GOTO 450
440 IF A(F(G))=Q THEN A(F(G))=X
450 NEXT G
460 A(N)=X
470 RETURN
500 V=54296:D=54277:R=54278:F=54276
510 POKE V,15:POKE D,36:POKE R,68:POKE F,17
520 POKE 54273,57:POKE 54272,172
530 FOR D=1 TO 100:NEXT
550 POKE 54273,45:POKE 54272,198
560 FOR D=1 TO 200:NEXT
570 POKE D,0:POKE F,0:POKE R,0
580 RETURN
```

HANGMAN

Hangman is a traditional word game which involves the guessing of letters which make up an unknown word. The computer selects a word from its DATA store (lines 315 to 370) and then tells you how many letters the word contains and how many guesses you have to discover the word.

If you correctly guess a letter, the computer will place the letter in the right position in the word. If guess wrongly the computer will deduct one from the number of guesses you have left.

The program relies on the store of words in its DATA statements. This program has a vocabulary of 100 words. You can easily increase your computer's word power by adding more DATA lines. The only other change which needs to be made is in line 50. The number contained in the random number generator must equal the number of words in the store. So just change the number 100 to equal the number of words in the DATA store.

Line 50 picks a random number from 1 to 100. Line 60 then READS down through the DATA statements until it reaches this word. For example if the computer picked 45, it would then READ through the DATA until it came to the 45th word, FINISHED, this word then becomes the word you are trying to guess. The program assigns this word to the variable A$. Line 90 then works out the length of A$ to find out how many letters it contains, this number is then assigned to the variable N. Line 110 uses MID$ to break the string A$ down into individual letters, they are converted to their ASCII code and stored in the B array. Line 220 converts your guess to its ASCII code so that it can be compared with the contents of the B array. This is done by lines 230 to 250. If it doesn't find a match, the program prints out a dash (line 420), if the letter has been found in the B array the computer then prints the letter in the correct place on the screen.

```
10 REM HANGMAN
20 PRINT "□":REM CLEAR SCREEN
30 Z=80:REM Z=NO. OF WORDS
35 POKE 53280,0:POKE 53281,14
40 DIM D(20):DIM B(20)
50 FOR G=1 TO INT(100*RND(1))+1
```

```
60 READ A$
70 NEXT
80 Y=0:W=0
90 N=LEN(A$)
100 FOR G=1 TO N
110 B(G)=ASC(MID$(A$,G,1))
120 D(G)=B(G)
130 NEXT
140 Q=N:PRINT TAB(10) "■■■■■■■■■■■■YOU HAVE";Q;
   "GUESSES":REM RED DOWN CURSOR
145 FOR DD=1 TO 1500:NEXT
150 FOR J=1 TO Q:Y=Y+1
160 GOSUB 360
170 IF H=N THEN 290
180 PRINT TAB(8) "YOU HAVE";Q+1-J;"GUESSES LEFT"
200 GOSUB 500
210 INPUT "■■■■■■■■■■NEXT GUESS";C$:REM RIGHT
    CURSOR DOWN
220 F=ASC(C$)
230 FOR G=1 TO N
240 IF D(G)=F THEN D(G)=0:J=J-1
250 NEXT:NEXT
260 GOSUB 410
270 PRINT TAB(4) "SORRY, TIME IS UP THE WORD WAS"
280 PRINT TAB(16) "■■" A$:GOTO 310
290 PRINT TAB(2) "WELL DONE YOU GUESSED THE WORD
    IN";Y-1
295 PRINT TAB(16) "■TRIES":REM CURSOR DOWN
310 FOR DD=1 TO 3000:NEXT:RUN
315 DATA COUGH,BOUGH,DRAUGHT,CAUGHT,WELCOME,
    CHECKED,ASSIGN,SHRUGGED,PATRONISE
320 DATA RECORD,DECODE,MIDDLE,FIDDLE,POWER,LOWER,
    DOUBLE,TROUBLE,LIGHTLY,PLIGHT
325 DATA FLIGHT,SIGHTED,FOUGHT,BOUGHT,SOUGHT,
    TROUGH,PERSONAL,ASSUME,PROGRAM
330 DATA POSTAGE,FOLLOW,WALLOW,CHALICE,MALICE,
    SCHEME,DISASTER,REACH,SEARCH
335 DATA EVER,WATER,WANTED,WAITED,WAITER,MINISTER,
    SINISTER,FINISHED,WIZARD
```

```
340 DATA LIZARD,WICKED,WIZENED,EVIL,WEEVIL,WHEELS,
    BLIZZARD,PARTICLE,ATOM
345 DATA ARTICLE,ELECTRON,STARTED,PARTED,FAMISHED,
    EAGLE,LEGAL,TENOR,FRANTIC
350 DATA TERRIER,BANANA,FIGURE,IDIOT,NARCOTIC,
    PATHETIC,TERROR,HORROR,POSTURE
355 DATA ELEPHANT,TRIUMPHANT,STATUS,BACHELOR,
    ANSWER,PACIFIC,PUBLICATION,HAWK
360 DATA NEGATIVE,CHANGES,EXPAND,VERBAL,CRITICIZE,
    SAMPLE,DECOY,BEYOND,MULTIPLY
365 DATA UMBRELLA,VERTIGO,TRAGIC,SENATOR,PSEUDO,
    NOISELESS,MUSICIAN,LITERATE
370 DATA GURGLE,COHESIVE,DISCORD,ECLIPSE,OUTRAGE,
    RELIC,NURSERY
410 H=0:PRINT "□":REM CLR SCRN
415 PRINT "▒▒▒▒▒":REM CURSOR DOWN
417 FOR E=1 TO N
420 IF B(E)=D(E) THEN PRINT TAB(16) "-";
430 IF B(E)<>D(E) THEN PRINT TAB(16) CHR$(B(E));
    :H=H+1
440 NEXT:PRINT
460 PRINT:IF H<>N THEN PRINT TAB(10)H;"CORRECT
    LETTERS"
480 PRINT
490 RETURN
500 POKE 54296,15
510 POKE 54277,34:POKE 54278,34
520 POKE 54276,33
525 POKE 54273,51:POKE 54272,97
530 FOR DD=1 TO 250:NEXT
540 POKE 54277,0:POKE 54278,0:POKE 54276,0
550 RETURN
```

BREAKOUT

In this version of the game Breakout, the object is to hit the bouncing ball with your bat and bounce it back up the screen to knock out the rows of hearts and get the ball through to the checkerboard pattern at the top of the screen. You start the game with ten balls and lose one each time you miss hitting it with the bat. The game will end if you miss all ten balls.

The "Z" key moves your bat to the left and the "N" key moves it to the right.

The graphic symbol in line 20 is obtained by holding down the Commodore key and pressing the plus sign (+) key. The heart symbol is on the "S" key and the open circle or ring is on the "W" key. These two are obtained by holding down the shift key.

Lines 70 and 80 set the starting position of the ball and line 82 chooses the starting position of the bat.

Line 120 keeps the ball from going off the left or right hand side of the playing area. Line 130 checks the position of the ball. When it is gone below line 21 on the screen, the program goes to the routine from line 2000 which checks to see if the ball has hit the bat. If you have missed, the program goes to lines 2500 onward where the score is printed up and a ball is deducted from the number you have left. Lines 2540 and 2550 select a new starting position for the next ball and line 2560 starts your bat off again. Line 2570 sends the program back to line 90 for the next turn.

If you have hit the ball, line 2002 calls up the routine which creates the sound effects and then line 2005 changes the direction of the ball to send it back up the screen. LIne 145 checks to see if the ball has reached the top of the display, if it has the game ends and you are congratulated.

Lines 140, 144 and 146 check to see if the ball has hit the other graphic symbols. Lines 150 and 160 increment the position of the ball on the screen.

Variable S is the screen location of the bat. Variable F relates to the movement of your bat on the screen. Each time you move the bat the value of F is changed. By adding F to the screen location the variable S is then used to POKE the bat onto the screen. The first two POKE statements in line 230 put a blank space either side of the bat. This is to erase previous screen locations of the bat.

```
5 REM BREAKOUT
10 GOSUB 5000
15 PRINT "■":REM CLEAR SCREEN
18 POKE 53280,0:POKE 53281,11
20 PRINT TAB(8) "■▒▒▒
```

```
144 IF PEEK(1024+40*BD+BA+1)=83 THEN GOSUB 3000:
    GOTO 150
145 IF BD<2 THEN GOTO 4000
146 IF PEEK(1024+40*BD+BA-1)=83 THEN GOSUB 3000
150 BA=BA+X
160 BD=BD+Y
200 GET A$
210 IF A$="Z" THEN F=F-1
220 IF A$="N" THEN F=F+1
222 IF F<8 THEN F=8
223 IF F>39 THEN F=39
225 S=1904+F
230 POKE S-1,32:POKE S+1,32:POKE S,111:
    POKE S+54272,0
500 GOTO 100
1000 GOSUB 3500:X=-X
1200 RETURN
2000 IF ABS(F-BA)>2 THEN 2500
2002 GOSUB 3500
2005 Y=-Y
2200 GOTO 140
2500 CX=CX-1:PRINT "■":REM HOME
2505 IF CX=0 THEN GOTO 2600
2510 IF CX>0 THEN PRINT "■SCORE":PRINT "■"SC
2515 IF CX>0 THEN PRINT "■BALLS":PRINT "■"CX
2518 FOR DD=1 TO 1000:NEXT DD
2520 FOR TT=1904 TO 1943:POKE TT,32:NEXT
2530 IF CX=0 THEN GOTO 4040
2540 BA=INT(6*RND(1))+14:REM START BALL ACROSS
2550 BD=10:REM START BALL DOWN
2560 F=13+INT(RND(1))*10:REM START OF BAT ACROSS
2570 GOTO 90
2600 FOR DD=1 TO 2000:NEXT
2610 PRINT "■":POKE 53280,4:POKE 53281,2
2630 PRINT "■■■■■■■" TAB(14) "■GAME OVER":REM YELLOW
2640 PRINT "■" TAB(12) "YOU SCORED":SC
2650 GOTO 4040
3000 GOSUB 3500:POKE 1024+40*BD+BA,32
3001 IF RND(1)>.95 THEN POKE 1024+40*BD+BA+1,32
3002 IF RND(1)<.1 THEN POKE 1024+40*BD+BA-1,32
3010 SC=SC+437
3020 PRINT "■■■SCORE":PRINT "■" SC
```

```
3035 Y=-Y
3040 RETURN
3500 POKE 54296,15
3510 POKE 54277,17:POKE 54278,17
3520 POKE 54276,17
3530 POKE 54273,28:POKE 54272,214
3540 FOR DD=1 TO 25:NEXT
3550 POKE 54276,0:POKE 54277,0:POKE 54278,0
3560 RETURN
4000 FOR DD=1 TO 2000:NEXT DD
4010 PRINT "■":POKE 53280,2:POKE 53281,6
4020 PRINT "■■■■■■■■■■" TAB(10) "■YOU HAVE SUCCEDED"
     :REM BLACK
4030 PRINT "■■" TAB(10) "■YOU SCORED"SC*CX:REM
     YELLOW
4040 FOR DD=1 TO 3000:NEXT DD
4050 RUN
5000 PRINT "■":REM CLEAR SCREEN
5010 POKE 53280,11:POKE 53281,0
5020 PRINT "■■■■■■■■■■"
5030 PRINT TAB(10) "■******* ■BREAKOUT ■*******":REM
     YELLOW RED
5040 FOR DD=1 TO 2500:NEXT
5050 RETURN
```

RED ALERT

You are the commander-in-chief of the space-ship APPOLLEX on a secret mission in outer space. It is your duty (as well as the computer's) to place various space modules in quadrants of space. The space modules that you each have available for each quadrant are as follows:-

6 Phasons. These are satellite bombs, and if hit will subtract 250 points from you or your computer's score.

6 Meetons. These are satellite starship-substations. If you or your opponent should land on one of these, the score is incremented by 300.

1 Starship. This is the most important module and if hit will increment the score by 2000.

You have to try and place the modules in each quadrant in positions so that your opponent will land on the Phasons and lose points.

The game is in two sections. The first section allows you to place your modules. This is done very easily by moving the cursor by hitting the cursor keys (right and down). When the cursor is positioned at the coordinate position that you want, you select the type of module by typing in the first letter of that particular module: 'P' for Phason, 'M' for Meeton and 'S' for Starship. The computer will be doing the same thing.

Once you have deposited all your modules in the quadrant the game moves onto the next section which is the red-alert section. You now take it in turns with the computer to choose and interrogate coordinates on the screen. This is done very easily by moving the cursor as before and hitting the RETURN key to interrogate a square. You are immediately shown whether you have had a hit or not. And your score is altered accordingly. The computer also has a turn after you, and the results are immediately relayed to the matrix on the screen.

Should you or your opponent hit a starship, the search for that quadrant is ended and you move onto the next quadrant. You are allowed a maximum of 20 coordinate interrogations per quadrant. If you exceed this number, the program will automatically take you to the next quadrant.

There is also a time limit of 10 minutes for the game. The game automatically ends at this point, and the scores are displayed. If you want to play again just strike the letter "A" for another game.

Warning. Your on board computer will not allow you to either place more than 6 modules in line (horizontally or vertically) or interrogate more than six positions in line (horizontally or vertically). If you do this, the computer will go into an indefinite loop and the only way to get out is with the STOP key.

Structure of the Program.

Lines 20-22. Define function FNR to equal a randomn number (X)0. X$ is a cursor positioning string.

Line 30. Cursor positioning string.

Line 40. DIMension arrays P%(7,7) and C%(7,7). P%(7,7) will hold the player's quadrant information (phasons, meetons, starship) and array C%(7,7) will hold the computer's quadrant information. Although the matrix on the screen is an 8 by 8 matrix, the arrays are dimensioned to allow use of the cells (0,0), saving memory space.

Line 50. Initialise the real time clock. Go to the subroutine (line 760) and initialise the variables PH, M, S, C. These variables hold the starting values for each module in a quadrant. The variable Q is incremented by 1. This is the quadrant number. Go to line 130 (set-up section).

Lines 60-120. Draw screen for red-alert section.

Lines 130-170. Draw screen for set-up section.

Line 180. Clear the arrays P%(7,7) and C%(7,7). Fill them with zeroes. The X and Y variables are used for screen cursor positioning. They are set to zero which is the top left hand corner of the players matrix. Go to subroutine 710 and position the cursor at the top left hand corner of the matrix. Now print a reverse '*'.

Line 190. Strobe keyboard for input. Go to a few nested subroutines and update the screen clock while waiting for input from the keyboard.

Line 200. We are about to move the cursor, so we check to see if the current cursor position and the matrix coordinates P%(X,Y) contain a character. If not, we print a "." in reverse.

Line 210. If a cursor key (csr-down) has not been struck then go to line 240.

Line 220. A cursor down key has been struck. We accordingly increment the value held in the variable Y by one. If Y equals 8 then we have gone past the screen location, so we place a 0 in the variable which will position the cursor at the far left position of the player's matrix display.

Line 230. If the x,y position in P%(X,Y) is not equal to zero then we must move the cursor one more position to the right. We accordingly go back to line 220 and increment the variable Y.

Lines 240-260. This contains the same logic as the previous three lines, only the X variable is incremented instead of the Y variable.

Lines 270. If A$ does not hold a "P" then GOTO line 320.

Line 280. A$ holds a "P". We now do an error check to make sure that the spot where the player wishes to deposit a phason is not occupied. If it is occupied, then we go back to line 190 and get another command.

Line 290. The current spot does not hold a module (phason, meeton, starship). We therefore decrement the variable PH by one. We check to see if all the phasons have been deposited on the screen, and if so we go to line 320.

Line 300. We go to the cursor positioning subroutine (line 71) and come back and print a "P" on the screen within the players displayed matrix. We also update the array P%(X,Y) with a 1. We now reposition the cursor and print the remaining phasons on the screen beside the word phason.

Line 310. We go to line 410 and check to see if all of the modules for the quadrant have been 'coorded' (phasons, meetons, starship).

Lines 320-360. The logic is similar to the previous five lines. The main difference being that we are comparing for the character "S" to be in the variable A$.

Line 410. We check to see if all of the modules (phason, meeton, starship) have been 'coorded' on the screen.

Line 420. We arrive here if an illegal key has been hit. We therefore go back to get another character.

Line 430. A cursor-only move has been made. We therefore print a reverse '*' and go back and get another command.

Line 440. We go to subroutine 740. This subroutine clears the computer matrix array C%(7,7) and then fills it with the various modules (phasons, meetons, starship). Having done that, we go to line 60 which is the set-up section.

Line 450. The player and computer have each checked a position in the matrix. We therefore increment the coordinate counter (variable C) and print the value of C on the screen. We now position the cursor at the top left hand position of the player's matrix (X=0, Y=0) and print a reverse '*'.

Line 460. We get a character from the keyboard. While we are waiting for a character we go and update the screen clock.

Line 470. If a carriage return has not been hit then we go to line 550.

Line 480. A carriage return has been hit. We therefore increment the coordinate counter and check to see if more than 20 coordinates have been entered for the present quadrant. If this is so, we go on to the next quadrant set-up section (go to 130).

Line 490. The 'number of coordinates' check for this quadrant is less than 200. We now reposition the cursor and print the new coordinate number.

Line 500. If the current computer coordinate position holds a starship (value 3) then we print 'S' in the relevant position on the player's matrix. We then increment the player's score by 2000 and go on to the next quadrant set-up section (go to 130).

Line 510. If the current computer coordinate position holds a phason (value 1) the we subtract 250 from the player's score and print at the player's matrix position a "P".

Line 520. If the computer's coordinate position holds a meeton (value 2) then we increment the player's score by 300 and print a "M" at the matrix position.

Line 530. If the computer's coordinate position holds nothing (value 0) then we print a "+" at the player's matrix position.

Line 540. We position the cursor at the player's score and wipe it clean. We then reposition the cursor and update the player's score.

Line 550. We branch to this line from line 470 only if a non-carriage-return character has been hit. We check the computer coordinate position and if the position holds nothing (value 0) we print a reverse "." at the player's matrix position.

Lines 560-620. Cursor positioning routines similar to those at lines 210-260.

Line 630. We go to subroutine 830. This subroutine temporarily stores the values of X1 and Y1 in X and Y. It is the computer's turn to pick a coordinate for interrogation so we go and get two random numbers for the coordinates' variables X and Y.

Lines 640-670. This is similar to lines 500-530, the difference being that we are checking to see if the computer has picked a coordinate that contains a module.

Line 680. Position cursor over computer's score. Rub out and then print computer's score.

Line 690. Line drawing subroutine called from the set-up and red-alert drawing routines.

Lines 700-720. Cursor positioning subroutines.

Line 730. Places two random numbers from 0-7 into the variables A and B. This subroutine is called when the computer sets up its quadrant and also when the computer is picking coordinate positions.

Lines 740-750. Clears the computer array C%(7,7) and fills it with the various modules.

Line 760. A new quadrant is now being entered. We therefore reinitialise the variables containing the number of modules available and return to the set-up section.

Line 770. Reposition the cursor and print the updated time in minutes and seconds.

Line 780. If the time for the game has exceeded 10 minutes then the game is over, so branch out of the program to line 800.

Line 780. The time limit has not been passed so we return to the calling routine.

Lines 800-820. The game is over, so we clear the screen, display the results, and await for the player to input a command to play again character "A".

Lines 830-840. Swap the X,Y, X1, and Y1 values.

Line 850. Delay routine used when a starship is found.

```
10 REM RED ALERT
12 GOSUB 1000
20 DEFFNR(X)=INT(RND(1)*X):PRINT"■";
22 X$="■■■
```

```
130 PRINT"■";TAB(13)"■■SETUP":PRINT:PRINTTAB(2)"
    ■■TIME";SPC(4);"QUADRANT"
131 PRINT
132 REM CLR SCRN REV ON YELLOW WHITE REV ON
135 IF Q>6 THEN 800
140 PRINTTAB(15);"■";Q:PRINT:PRINTTAB(10)"
    ■■PLAYER"
145 REM REV ON YELLOW
150 PRINT "■┌";:GOSUB 690:PRINT "┐":FORI=1TO8
155 REM WHITE REV ON REV OFF
157 PRINT "│■.......■│":NEXT
160 PRINT "└";:GOSUB690:PRINT"┘":X=12:Y=10:GOSUB700
162 PRINTTAB(15)"■■P■HASON■■";PH
165 REM REV ON LT RED YELLOW WHITE REV OFF
170 Y=12:GOSUB700:PRINTTAB(15)"■■M■EETON■■";M:Y=14:
    GOSUB700
172 PRINTTAB(15)"■S■TARSHIP■";S
175 REM REV ON GREEN YELLOW REV OFF REV ON CYAN
    WHITE REV OFF
180 FORX=0TO7:FORY=0TO7:P%(X,Y)=0:C%(X,Y)=0:NEXT:
    NEXT:X=0:Y=0
185 GOSUB 710:PRINT"■■";
187 REM REV ON
190 GETA$:PRINT"■";:GOSUB 770:IF A$="" THEN 190
200 IFP%(X,Y)=0THEN GOSUB710:PRINT "■.";
205 REM REV ON
210 IFA$<>"■"THEN 240
215 REM CRSR DOWN
220 Y=Y+1:IFY=8THENY=0
230 IFP%(X,Y)THEN220
240 IFA$<>"■"THEN270
245 REM RIGHT CRSR
250 X=X+1:IFX=8THENX=0
260 IFP%(X,Y)THEN250
270 IFA$<>"P"THEN320
280 IFP%(X,Y)THEN190
290 PH=PH-1:IFPH<0THENPH=0:GOTO320
300 GOSUB710:PRINT"■■P■";:P%(X,Y)=1:GOSUB830:X=12
    :Y=10:GOSUB700
```

```
301 PRINT SPC(10)PH
302 REM REV ON LT RED WHITE REV OFF
305 GOSUB 840
310 GOTO410
320 IF A$<>"M" THEN 370
330 IF P%(X,Y)THEN 190
340 M=M-1:IF M<0 THEN M=0:GOTO 370
350 GOSUB710:PRINT"■■1■";:P%(X,Y)=2:GOSUB830:X=12
    :Y=12:GOSUB700:PRINTSPC(10)M;
352 REM REV ON GREEN WHITE REV OFF
355 GOSUB 840
360 GOTO 410
370 IF A$<>"S" THEN 430
380 IF P%(X,Y) THEN 190
390 S=S-1:IF S<0 THEN S=0:GOTO 430
400 GOSUB710:PRINT"■S■";:P%(X,Y)=3:GOSUB830:X=14
    :Y=14:GOSUB700:PRINTSPC(10)S
402 REM REV ON CYAN WHITE REV OFF
405 GOSUB 840
410 IF PH=0 AND M=0 AND S=0 THEN 440
420 GOTO 190
430 GOSUB710:PRINT"■";:GOTO 190
435 REM REV ON
440 GOSUB 740:GOSUB 60:
450 C=C+1:X=10:Y=4:GOSUB700:PRINT;C:X=0:Y=0:GOSUB
    710:PRINT"■";:REM REV ON
460 GETA$:PRINT"■";:GOSUB770:IFA$="" THEN 460:REM
    REV ON
470 IFA$<>CHR$(13)THEN550
480 C=C+1:IFC>20THENGOSUB760:GOSUB850:GOTO130
490 GOSUB830:X=10:Y=4:GOSUB700:PRINTC;:GOSUB840
500 IFC%(X,Y)=3THEND=D+2000:C%(X,Y)=98:GOSUB710:
    PRINT"■S";:GOSUB760:GOSUB850
502 REM REV ON CYAN
505 IFC%(X,Y)=98 GOTO 130
510 IFC%(X,Y)=1THEND=D-250:C%(X,Y)=N:GOSUB710:
    PRINT"■■P■";
515 REM REV ON LT RED REV OFF WHITE
520 IFC%(X,Y)=2THEND=D+300:C%(X,Y)=N:GOSUB710:
    PRINT"■■1■";
```

```
525 REM REV ON GREEN REV OFF WHITE
530 IFC%(X,Y)=0 THEN C%(X,Y)=N:GOSUB710:PRINT"+";
535 REM USE SHIFT AND PLUS KEY
540 GOSUB830:X=3:Y=19:GOSUB700:PRINT"■          ■";
    :GOSUB700:PRINT;D;:GOSUB840
542 REM REV OFF REV ON
545 GOTO 630
550 IFC%(X,Y)<>NTHENGOSUB710:PRINT"■.";
555 REM REV ON
560 IFA$<>"■"THEN590:REM CRSR DOWN
570 Y=Y+1:IFY=8THENY=0
580 IFC%(X,Y)=NTHEN570
590 IFA$<>"■"THEN620:REM RIGHT CRSR
600 X=X+1:IFX=8THENX=0
610 IFC%(X,Y)=NTHEN600
620 GOSUB710:PRINT"■■■";:GOTO460:REM REV ON WHITE
630 GOSUB830:GOSUB730:X=A:Y=B:FORI=1TO1500:NEXT
640 IFP%(X,Y)=3THENE=E+2000:GOSUB720:PRINT"■S";:
    GOSUB760:GOSUB850:GOTO130
645 REM REV ON CYAN
650 IFP%(X,Y)=1THENE=E-250:P%(X,Y)=N:GOSUB720:
    PRINT"■P";:REM REV ON LT RED
660 IFP%(X,Y)=2THENE=E+300:P%(X,Y)=N:GOSUB720:
    PRINT"■M";:REM REV ON GREEN
670 IFP%(X,Y)=0THENP%(X,Y)=N:GOSUB720:PRINT"X":REM
    USE V KEY WITH SHIFT
680 X=13:Y=19:GOSUB700:PRINT"■          ■";:GOSUB700
    :PRINT;E;:GOSUB840:GOTO460
685 REM REV OFF WHITE REV ON
690 FORI=1TO8:PRINT"─";:NEXT:RETURN:REM USE SHIFT
    AND C KEY
700 PRINTLEFT$(X$,X);LEFT$(Y$,Y);:RETURN
710 PRINTLEFT$(X$,X+OX);LEFT$(Y$,Y+OY);:RETURN
720 PRINTLEFT$(X$,X+XC);LEFT$(Y$,Y+OY);:RETURN
730 A=FNR(8):B=FNR(8):RETURN
740 FORA=0TO7:FORB=0TO7:C%(A,B)=0:NEXTB:NEXTA:
    FORI=1TO2
750 FORJ=1TO6:GOSUB730:C%(A,B)=I:NEXTJ:NEXTI:GOSUB
    730:C%(A,B)=3:RETURN
```

```
760 PH=6:M=6:S=1:C=0:Q=Q+1:RETURN
770 GOSUB830:X=2:Y=4:GOSUB700:PRINTMID$(TI$,3,2);"
    :";RIGHT$(TI$,2);:GOSUB 840
780 IFTI$>"001000"THEN800
790 RETURN
800 PRINT"□■PLAYER";D:PRINT:PRINT:PRINT")
    (COMPUTER";E
810 GETA$:IFA$<>"A"ORA$=""THEN81
820 CLR:RUN
830 X1=X:Y1=Y:RETURN
840 X=X1:Y=Y1:RETURN
850 FORI=1TO3000:NEXT
860 PRINT"□"
870 PRINT:PRINT:PRINT:PRINT:PRINT
880 PRINTTAB(12) "■PLAYER";TAB(20)"■COMPUTER"
890 PRINT:PRINTTAB(12);D;TAB(22)E
900 FOR I=1 TO 5000:NEXT:RETURN
1000 PRINT"□"
1005 POKE 53280,0:POKE 53281,2:PRINT"■"
1010 PRINT TAB(14)"■■■■■■■■■■■■■RED-ALERT"
1020 GOSUB 2000
1030 FOR I=1TO500:NEXT
1040 POKE 53280,2:POKE53281,0
1050 GOSUB 2000
1060 FORI=1TO500:NEXT
1070 TL=TL+1:IFTL=5THENRETURN
1080 GOTO 1005
2000 POKE 54296,15:POKE 54277,66:POKE54278,68
2010 POKE 54276,33
2020 POKE 54273,34:POKE 54272,75
2030 FORI=1TO500:NEXT
2040 POKE 54276,0:POKE 54277,0:POKE 54278,0
2050 RETURN
```

SYMBOL MATCH

Here it is, folks. A game which requires 100% unadulterated skill to play. This is the Commodore 64 version of the well-known game of Concentration.

The computer will randomly select 18 pairs of different coloured characters, and hide them behind 36 positions on the screen. Each player takes it in turns displaying two characters at a time. If a match occurs then the player's score is incremented.

Moving around the screen is easy, as the only controls used are the cursor controls and the RETURN key which is used to interrogate a square.

A word of warning: The RETURN key is very sensitive. If you hit this key over and over again rapidly, the program will crack up as it won't be able to stand the pace!

Structure of the program

Lines 50-60. String cursor positioning strings.

Line 80. Define function FNR to equal a random number >1 <=X.

Line 90-100. DIMensioning of arrays and initialisation of variables. Array A%(1-36) will ultimately hold the 16 pairs of random characters (POKE characters values). Array B%(1-36) holds the initial array of POKE characters (ordered). Array SCR%(1-36) will hold the screen POKE values for the 36 different positions on the screen. The array DOT%(1-36) will hold the alphanumeric values 0-9-A-Z, which are displayed on the screen at the start of the game. This array is continuously being updated whenever a match occurs the two squares are filled with a space (Poke value 32). This value is placed in the corresponding cell within the DOT%(1-36) array. The arrays C11%(1-36) and CL2%(1-36) will hold the random colors (POKE values) for the 36 different graphic charactes. Array CL1%(1-36) will ultimately hold the randomised colors. Variable P1 is the first player's score holder. Variable P2 is the second player's score holder. Variable 0 holds the offset for the color screen map (screen + offset).

Lines 110-140. Display title and draw board.

Line 150. This line together with the subroutine called sets up the SCR%(1-36) array with the POKE coordinates for the displaying the graphic symbols on the screen.

Lines 160-170. The DOT%(1-36) array is initialised with the appropriate POKE values for the alphanumberics 0-9-A-Z.

Lines 180-190. The arrays CL1%(1-36) and CL2%(1-36) are filed with the POKE values for the colors of the graphic characters.

Line 200. POKE to the screen the alphanumerics held in the array DOT%(1-36).

Line 210. The array B%(1-36) is filled with the 16 pairs of POKE graphic values which are held in the DATA statement at line 270.

Lines 210-220. The arrays A%(1-36) and C2L%(1-36) are filled with sixteen random characters and colors taken from arrays B%(1-36) and CL2%(1-36).

Line 240-250. Print "PLAYER1=",PLAYER2=".

Line 270. POKE graphics data.

Line 280. Subroutine called from line 150. This routine fills SCR%(1-36) with POKE values for the 36 different screen positions.

Line 290. The variable R will hold the screen position value for the square that is being interrogated. The screen positions increment in value from 1 (upper left hand corner) to 36 (bottom right hand corner).

Line 310. Go to screen position checking routine.

Line 320. Check the variable R (position of square being referenced on screen) with the number in R%(1). If the numbers are equal then we know that the square being referenced has already been selected as one of the two choices made by the player. We therefore move the cursor one position to the right (R=R+1) and go to the subroutine (starting line 890) and do some further checking.

Line 330. Check if the first square (number one) has already been picked. Go to subroutine and do some more checking.

Line 340. POKE the screen position currently being referenced with the reverse character held in array DOT%(1-36).

Line 350. Call GET$ routine for input of any key from the keyboard.

Lines 360-390. Check contents of A$ with the four cursor control keys (csr-down, csr-up, csr-right, csr-left) and give variable M a value equal to the ASCII value of A$.

Line 400. POKE the square being referenced with the normal character (not reversed) held in array DOT%(1-36).

Lines 410-500. Conditionals which are acted upon depending on the cursor control key that was struck. For example if M=29, then a csr-right key has been struck, so we increment the variable R by 1.

Lines 510-520. Check to see whether the square being referenced either contains a blank space (POKE character 32) or has already been picked as one of the two choices (R=R%(1)). If either of these conditions exist, then we go back and move the cursor again (lines 410-500).

Line 530. Return from subroutine. Two nested subroutines are involved. The GOSUB at line 580 which is invoked at line 310 calls a further subroutine GOSUB890.

Line 540. Both of these variables (F and T) are used as counters to tell the computer whose turn it is.

Lines 550-570. Check whose turn it is, and POKE a yellow ball beside the player who is choosing a square.

Line 580. Go to the subroutine and check if a cursor key has been struck.

Line 590. Short delay.

Line 600. Go to screen position subroutine.

Line 610. Rub out the yellow ball.

Line 620. If the character held in A$ is not a carriage return (CHR$(13)) then GOTO line 670.

Lines 630-660. A carriage return has been hit, so we now display the character underneath the cursor and its color.

Lines 670-680. If either Z=1 or T is not equal to 3 then goto 740.

Line 700. If both squares displayed on the screen are the same (T%(1)=T%(2)) then we have a match. Player one's score is incremented by one (P1=P1+1), the total number of pairs is incremented by one (TT=TT+1), the two screen positions are filled with spaces (DOT%(R%(1))=32, DOT%(R%(2))=32), and the X and Y variables are given the screen coordinates of the first player's score. We now go to the subroutine to position the cursor beside the first player's score (subroutine 870).

Line 710. If a match has occurred, we update the first player's score.

Line 720. POKE the two squares just referenced with white.

Line 730. POKE the two squares last referenced with the contents of DOT%(1-36). Place two dummy numbers in R%(1) and R%(2).

Line 740. If the variable T is not equal to 5 then GOTO line 800.

Line 760. POKE the two squares last referenced with white.

Line 770. If the two squares last referenced are the same then we have a match. Update player two's counter (P2=P2+1), and go to subroutine to position the cursor beside player two's score.

Line 780. If there is a match then update (on the screen) player two's score.

Line 790. POKE the two squares last referenced with the contents of DOT%(1-36).

Line 800. Check if the flag (variable FL) equals 3 and if so change its value to a 1.

Line 810. If the value R (square being referenced value 1-36) is larger than 36 then let the value equal the top left hand corner (R=1).

Line 820. If the variable T holds a 5 then each player has had two choices. We therefore go back to line 54 and let each player choose again.

Line 830. If the key just hit was a carriage return (CHR$(13)), then increment variable R by 1 and give variable FL a value of 1.

Line 840. Go to the screen position checking routine.

Line 850. If the key just struck was a carriage return then increment the turn counter (variable T) by 1. Go to line 550.

Line 860. We arrive here if only a cursor movement has been given. We now go back to line 550.

Line 870. Cursor positioning subroutine.

Line 880. If the number of matches counter equals, or is larger than 18 then the game is over.

Lines 890-940. Screen position checking subroutine.

Line 970-980. Wait for a key to be hit. As soon as a key is hit, then run another game.

```
10 REM SYMBOL MATCH
20 GOSUB 1500
40 POKE 53281,11:POKE 53280,6:PRINT "■":REM WHITE
50 X$="■■■
```

```
135 REM 10 SPACES H KEY 2 SPACES H KEY SPACE N KEY
137 REM AFTER ENTERING LINE 130 MOVE CRSR BACK INTO
    LINE BEFORE PRESSING RETURN
140 PRINT"          ";:FORI=1TO18:PRINT"■";:NEXT
145 REM 10 SPACES Y KEY
150 C=1:S=1115:FORU=1TO6:GOSUB280:S=S+102:NEXT
160 C=48:FORI=1TO36:IFC=58 THEN C=1
170 DOT%(I)=C:C=C+1:NEXT
180 C=1:FORI=1TO6:FORJ=1TO6:C1L%(C)=I:C2L%(C)=I:
    IFI=6THENC1L%(C)=7:C2L%(C)=7
185 IF I=2 THENC1L%(C)=10:C2L%(C)=10
190 C=C+1:NEXT:NEXT
200 FORI=1TO36:POKESCR%(I),DOT%(I):NEXT
210 FORI=1TO36STEP2:READN:B%(I)=N:B%(I+1)=N:NEXT
220 C=1:FORJ=36TO1STEP-1:R=INT(RND(1)*J+1)
230 A%(C)=B%(R):C1L%(C)=C2L%(R):C2L%(R)=C2L%(J):
    B%(R)=B%(J):C=C+1:NEXT:PRINT
240 PRINTTAB(12)"■■PLAYER■1■=":REM CYAN REV ON
    RIGHT CRSR
250 PRINT:PRINTTAB(12)"■PLAYER■2■=■■";
260 GOTO 290
270 DATA 65,66,67,81,88,86,87,83,90,91,95,102,105,
    108,109,113,114,127
280 FORI=1TO6:SCR%(C)=S:S=S+3:C=C+1:NEXT:RETURN
290 R=1
300 GOTO 540
310 GOSUB 890
320 IFR=R%(1)THENR=R+1:GOSUB890
330 IFR=1ANDR%(1)=1THENR=R+1:GOSUB890
340 POKESCR%(R),DOT%(R)+128:FOR DD=1 TO 50:NEXT
350 GET A$
360 IF A$="■" THEN M=17:REM DOWN CRSR
370 IF A$="■" THEN M=29:REM RIGHT CRSR
380 IF A$="■" THEN M=145:REM UP CRSR
390 IF A$="■" THEN M=157:REM LEFT CRSR
400 POKE SCR%(R),DOT%(R)
410 IF M=17 THEN R=R+6
420 IF R=42 AND M=17 THEN R=1
430 IF R>36 AND M=17 THEN R=R-35
440 IF M=29 THEN R=R+1
```

```
450 IF R>36 AND M=29 THEN R=1
460 IF M=157 THEN R=R-1
470 IF R=0 AND M=157 THEN R=36
480 IF M=145 THEN R=R-6
490 IF R<-4 AND M=145 THEN R=36
500 IF M=145 AND R<1 THEN R=R+35
510 IF DOT%(R)=32 THEN 410
520 IF R=R%(1) THEN 410
530 RETURN
540 FL=1:T=1
550 IF T<3 THEN PP=1834:GOTO 570
560 IF T>2 THEN PP=1914
570 POKE PP+0,7:POKE PP,81
580 M=0:GOSUB 310
590 FOR D=1 TO 100:NEXT
600 GOSUB 890
610 POKE PP,32
620 IF A$<>CHR$(13) THEN 670
630 IF R=1 AND R%(1)=R THEN R=R+1
635 GOSUB 1000
640 IF R+1=R%(1) THEN Q=1
650 POKE SCR%(R),A%(R):POKE SCR%(R)+0,C1L%(R):
    T%(FL)=A%(R):R%(FL)=R
660 IF Q=1 THEN R=R+1:Q=0
670 IF Z=1 THEN 740
680 IF T<>3 THEN 740
700 IF T%(1)=T%(2) THEN P1=P1+1:TT=TT+1:DOT%(R%(1))
    =32:DOT%(R%(2))=32:X=23:Y=20
705 IF T%(1)=T%(2) THEN GOSUB 870
710 IF T%(1)=T%(2) THEN PRINT P1;
720 POKE SCR%(R%(1))+0,1:POKE SCR%(R%(2))+0,1
730 POKESCR%(R%(1)),DOT%(R%(1)):POKESCR%(R%(2)),
    DOT%(R%(2)):Z=1:R%(1)=40
735 R%(2)=41
740 IF T<>5 THEN 800
760 POKE SCR%(R%(1))+0,1:POKE SCR%(R%(2))+0,1
770 IF T%(1)=T%(2) THEN P2=P2+1:TT=TT+1:DOT%(R%(1))
    =32:DOT%(R%(2))=32:X=23:Y=22
775 IF T%(1)=T%(2) THEN GOSUB 870
780 IF T%(1)=T%(2) THEN PRINT P2;
```

```
790 POKESCR%(R%(1)),DOT%(R%(1)):POKESCR%(R%(2)),
    DOT%(R%(2)):R%(1)=40:R%(2)=41
800 IF FL=3 THEN FL=1
810 IF R>36 THEN R=1
820 IF T=5 THEN Z=0:GOTO 540
830 IF A$=CHR$(13) THEN R=R+1:FL=FL+1
840 GOSUB 890
850 IF A$=CHR$(13) THEN T=T+1:GOTO 550
860 GOTO 550
870 PRINT LEFT$(X$,X);LEFT$(Y$,Y);:RETURN
880 IF TT>18 THEN 970
890 IF R>36 THEN R=1
900 IF DOT%(R)=32 THEN R=R+1
910 IF TT=18 THEN 970
920 IF R>36 THEN R=1:GOTO 900
930 IF DOT%(R)=32 THEN 890
940 RETURN
970 GET A$:IF A$="" THEN 970
980 CLR:RUN
1000 POKE 54296,15
1010 POKE 54277,17:POKE 54278,132:POKE 54276,17
1030 POKE 54273,17:POKE 54272,37
1040 FOR DD=1 TO 250:NEXT DD
1050 POKE 54276,0:POKE 54277,0:POKE 54277,0
1060 RETURN
1500 PRINT"■":REM CLEAR SCREEN
1510 POKE 53280,2:POKE 53281,6
1520 POKE 781,12:POKE 782,13:SYS 65520:PRINT
     "■SYMBOL - MATCH"
1530 GOSUB 1800
1540 POKE 781,12:POKE 782,13:SYS 65520:PRINT
     "■■SYMBOL - MATCH"
1550 GOSUB 1800
1555 TR=TR+1
1560 IF TR=4 THEN TR=0:RETURN
1570 GOTO 1520
1800 POKE 54296,15:POKE 54277,33:POKE 54278,152:
     POKE 54276,65
1810 POKE 54275,100:POKE 54274,15
1820 POKE 54273,28:POKE 54272,214
```

```
1830 FOR DD=1 TO 300:NEXT DD
1840 POKE 54276,0:POKE 54277,0:POKE 54278,0
1850 FOR DD=1 TO 500:NEXT DD
1860 RETURN
```

AL-KHWARIZMI

In this brain game, each player tries to match a sequence of single digit numbers (1-9) which are displayed on a 3 by 3 matrix on the screen. The game starts with two numbers in the sequence and on each correct entry by the player, builds the sequence up until a player makes a mistake.

When this happens the player is awarded a score equal to the number of numbers last guessed, and then the second player is given the opportunity of completing the sequence correctly. If both players miss the sequence, the number sequence is randomised and starts back at two numbers.

The winner is the first person to score 100.

Structure of the Program.

Line 60-70. Cursor positioning strings. BL$ holds four blank spaces (used for rubbing out).

Line 80. Define a function FNR to produce a random number >0 <10.

Line 100. DIMension arrays. The array NU%(1-100) will hold 100 random numbers (between 1-9). The array X%(1-13), Y%(1-13), will hold the x,y cursor positions for the screen. The variable CO is used as a counter to indicate the number of numbers that are currently in the sequence. Variable P1 holds the first player's score and variable P2 holds the second player's score.

Line 110. Go to the screen drawing subroutine for drawing the board.

Line 120. Read into array NU$(1-9) the nine alphanumeric numbers (1-9) together with there control color commands. The data statements for these numbers are found at line 81.

Line 130. Read into the arrays X%(1-13), Y%(1-13) the x and y cursor positions. The data statements for these positions are found at line 820.

Line 150. Fill the array NU%(1-100) with 100 random numbers (1-9).

Line 170. The variable I is used as the reference point from within the arrays X%(I) and Y%(I) to indicate the Ith element for the cursor positioning subroutine at line 570. PRINT T$ will print the word "TURN" in white underneath the player taking the turn.

Line 180. Go to the subroutine to display a random number on the screen.

Line 190. The variable I holds 1 which references the first element in the arrays X%(1-13), Y%(1-13).

Line 200. Wait for a character from the keyboard.

Line 210. If the character is non-numeric (ASCII=0) then go back and try again.

Line 220. If the numeric value is not equal to the number in the sequence then go to line 260.

Line 230. Display the number on the screen. Play a little music, GOSUB 880. Now rub out the number just displayed.

Line 240. If I>CO then the player has successfully completed the sequence. We therefore go and add one more number to the sequence and display it. We now go back and await for the player to input his or her guess, GOTO 180 (nested subroutine ending up at line 190).

Line 250. A non-numeric key has been struck, so we go back and try again.

Lines 270-360. These lines are used to check the input of player 1. If the number is not a match then the second player routine is invoked THEN 370 (line 330). If the first player is correct, then the score is incremented and he or she is allowed to try again.

Line 380. The second player's score is updated. An imporant condition is tested here (IF P1>99 OR P2>99 THEN 700). This checks to see if either of the players' scores is more than 99. If this is so the game is over, so we go directly to line 700 and display the results.

Line 390. The variables X1 and X2 hold the numbers in the sequence being tested. The numbers held in these variables is added to each player's score to increment the score. The second player does not have the sequence correct, so we reset X2 to equal zero and go and

play some music (GOSUB 960).

Line 400. The variable CO is now reset to 1 indicating that there is one number in the sequence. We now go to line 150 where the array NU%(1-100) is filled with 100 random numbers.

Lines 410-550. Initial screen display subroutine for drawing the graphics on the screen. This subroutine is called from line 110.

Line 550. Subroutine for drawing reverse spaces on the screen.

Lines 880-950. These lines play the fanfare each time a number is displayed on the screen.

Lines 960-100. These lines play a tone to indicate an incorrect number.

The rest of the lines are fairly self-explanatory.

```
10 REM AL-KHWARIZMI
20 VO=54296:AD=54277:SR=54278:WA=54276:HF=54273:
   LF=54272:HP=54275:LP=54274:TR=0
40 POKE 53280,5:POKE 53281,11:PRINT "■■";:REM
   CLEAR SCREEN WHITE
50 T$="■■TURN":REM REV ON WHITE
60 X$="■■■
```

```
210 X=I:IF VAL(A$)=0 THEN 200
220 IF VAL(A$)<>NU%(I) THEN 260
230 F=VAL(A$):I=F:GOSUB 570 :PRINT NUM$(F);:GOSUB
    880:GOSUB 570:PRINT "■ ";:I=X
235 REM REV OFF
240 I=I+1:IFI>COTHENGOSUB620:X=I:I=10:GOSUB570:
    PRINT"■";X-1:I=X:X1=X-1:GOTO180
245 REM REV ON YELLOW
250 GOTO 200
260 REM UPDATE PLAYER 1
270 X=I:I=12:GOSUB570:PRINT BL$;:P1=P1+X1:I=10:
    GOSUB570:PRINT"■";P1
275 REM REV ON YELLOW
280 X1=0:GOSUB 960
290 I=13:GOSUB 570:PRINT T$
300 CO=CO-1:GOSUB 620:GOSUB 670:I=1
310 GET A$:IF A$="" THEN 310
320 X=I:IF VAL(A$)=0 THEN 310
330 IF VAL(A$)<>NU%(I) THEN 370
340 F=VAL(A$):I=F:GOSUB570:PRINT NUM$(F);:GOSUB880
    :GOSUB570:PRINT"■ ";:I=X
345 REM REV OFF
350 I=I+1:IFI>COTHENGOSUB620:X=I:I=11:GOSUB570:
    PRINT"■";X-1:I=X:X2=X-1:GOTO357
352 REM REV ON LT RED
355 GOTO 360
357 GOSUB 670:I=1
360 GOTO 310
370 REM UPDATE PLAYER 2
380 X=I:I=13:GOSUB570:PRINT BL$:P2=P2+X2:I=11:
    GOSUB570:PRINT"■";P2
382 REM REV ON LT RED
385 IF P1>50 OR P2>50 THEN 700
390 X2=0:GOSUB 960
400 CO=1:GOTO 150
410 PRINT TAB(12) "■AL-KHWARIZMI":PRINT
420 GOSUB 550
430 FOR I=1 TO 3
440 PRINT TAB(13) "■ ■■■
```

```
445 REM REV ON REV OFF £, T AND * KEYS
450 PRINT TAB(13) "■ ■   ▌  ▌   ■ "
455 REM REV ON REV OFF G KEY
460 PRINT TAB(13) "■ ▀▀ ▛▀▀ ▛▀▀ ▛  "
465 REM REV ON REV OFF £ AND * KEYS
470 NEXT
480 GOSUB 550
490 PRINT:PRINT
500 PRINT TAB(8) "■PLAYER #1";SPC(3);"■PLAYER #2"
    :REM REV ON YELLOW LT RED
510 PRINT TAB(10) "■┌─────┐";SPC(5);" ┌─────┐":REM
    WHITE A, C AND S KEYS
520 PRINT TAB(10) "|";SPC(4);"|";SPC(5);"|";SPC(4)
    ;"|":REM B KEY
530 PRINT TAB(10) " └─────┘";SPC(5);" └─────┘":REM Z,
    C AND X KEYS
540 RETURN
550 PRINT TAB(13) "■              ":RETURN:REM REV ON
560 REM CURSOR POSITIONING
570 PRINT LEFT$(X$,X%(I));LEFT$(Y$,Y%(I));:RETURN
610 N=FNR(9):RETURN
620 FOR A=1 TO 500:NEXT:RETURN
630 FOR A=1 TO 100:NEXT:RETURN
670 I=1:CO=CO+1
680 FOR J=1 TO CO:I=NU%(J)
690 GOSUB570:PRINTNUM$(NU%(J));:GOSUB880:GOSUB570:
    PRINT"■ ";:GOSUB630:NEXT
695 RETURN
700 REM WE HAVE A WINNER
710 GET A$
720 IF P1>P2 THEN I=10:GOSUB 570:PRINT"■■";P1:I=12
    :GOSUB570:PRINT"■■WINS"
725 REM LT RED REV ON REV ON WHITE
730 IF P2>P1 THEN I=11:GOSUB570:PRINT"■■";P2:I=13
    :GOSUB570:PRINT"■■WINS"
735 REM REV ON YELLOW REV ON WHITE
740 I=FNR(9):GOSUB 570:PRINT NUM$(I):GOSUB 880
750 GOSUB 570:PRINT "■ ";:REM REV ON
760 IF P1>P2 THEN I=10:GOSUB 570:PRINT "■    ";
```

```
770 IF P2>P1 THEN I=11:GOSUB 570:PRINT "■    ";
775 REM WHITE
780 IF A$="" THEN 710
790 CLR:RUN
800 REM NUMBER DATA
810 DATA "■■1","■■2","■■3","■■4","■■5","■■6","■■7",
    "■■8","■■9"
820 REM X,Y DATA
830 DATA 16,4,19,4,22,4,16,7,19,7,22,7,16,10,19,10,
    22,10,12,17,23,17
835 DATA 12,20,23,20
880 POKE VO,15:POKE AD,30:POKE SR,132:POKE HP,60:
    POKE LP,20:POKE WA,65
890 POKE HF,25:POKE LF,177:FOR DD=1 TO 25:NEXT:
    GOSUB 950
900 POKE HF,28:POKE LF,214:FOR DD=1 TO 50:NEXT:
    GOSUB 950
910 POKE HF,43:POKE LF,52:FOR DD=1 TO 25:NEXT:
    GOSUB 950
920 POKE HF,34:POKE LF,75:FOR DD=1 TO 100:NEXT:
    GOSUB 950
930 TR=TR+1:IF TR=2 THEN TR=0:POKE WA,0:POKE AD,0
    :POKE SR,0:RETURN
940 GOTO 890
950 POKE HF,0:POKE LF,0:RETURN
960 POKE VO,15:POKE AD,17:POKE SR,132:POKE WA,33
970 POKE HF,5:POKE LF,103:FOR DD=1 TO 350:NEXT:
    GOSUB 1000
980 POKE HF,4:POKE LF,12:FOR DD=1 TO 500:NEXT:
    GOSUB 1000
990 POKE WA,0:POKE AD,0:POKE SR,0:RETURN
1000 POKE HF,0:POKE LF,0:FOR DD=1 TO 150:NEXT:
    RETURN
```

THE APPENDICES

Menu selection by cursor controls
Menu selection by function key
Timing input from the keyboard
Fast cursor positioning
Diet Helper
Runs per Over
Creating your own Games

MENU SELECTION BY CURSOR CONTROLS

This small subroutine allows the user to select a menu item from a list that is on the screen. Selection is made simple by requiring the user only to use the cursor-down key to position the cursor beside the menu item required. Once the cursor has been positioned, entry is made by hitting the RETURN key. This routine is ideal for anyone who is not accustomed to the Commodore 64 keyboard.

How it works.

The screen consists of 25 rows and 40 columns. For this discussion we will only be considering the rows. A numeric variable ROW is used which will tell the computer which row is currently being referenced. When the RETURN key is hit an ON ROW GOTO routine is used to send the computer to the particular menu that is wanted.

Row selection is made by using the cursor-down key. Each time this key is hit the value in ROW is incremented by one and the cursor is moved down by one row. A checking routine is required to make sure that the row number is not higher than the number of menu items on the screen. If the cursor reaches the bottom of the list, it is sent to the top of the list by a HOME command.

Assuming that a list of menu items has been sent to the screen, here is the subroutine that is required to choose any menu item in the list:-

The variable ROW has been initialised to equal 1 (first menu item).

```
230 PRINT"(HOME)";CHR$(X);
```

The cursor is sent to the top left of the screen and a character, where X = the value of any displayable character, is printed.

```
240 GET A$: IF A$="" THEN 240
```

We scan the keyboard for an entry by the user. The program will wait at this line until a single keyboard character is entered at the keyboard.

```
250 IF ASC(A$)=13 THEN ON ROW GOTO X1,X2,X3,XN
```

The computer checks to see if the RETURN key has been hit ASCII value 13. The variable ROW now holds a value equal to the item number in the list. The program now jumps to the Xnth value of ROW where Z1-XN are line numbers from 1 to the highest number available in ROW. For example if there where 5 menu items to choose from on the screen, then there would be five line numbers to GOTO.

```
260 IF ASC(A$)=17 THEN ROW=ROW+1
```

If the RETURN key has not been hit, the computer checks to see if the CURSOR-DOWN key has been hit ASCII value 17. If this is so then the row number counter ROW is incremented by 1.

```
270 IF  ROW =  L THEN PRINT "(LEFT) (HOME)"; CHR$(X);:
ROW=1: GOTO 240
```

If the value in variable ROW is one more than the number of items in the list then we have gone through the list and should place the cursor back to the start of the list. The computer first rubs out the display prompt by (LEFT) (SPACE) and then positions the cursor in the HOME position. The row counter variable ROW is now given a value of 1 for first row and the program goes back to get another character from the keyboard.

```
280    IF     ASC(A$)=17    THEN     PRINT     "(LEFT)
(DOWN)(LEFT)";CHR$(X);
```

If the cursor-down key has been hit then we rubout out the current prompt and move the cursor down one row.

```
290 GOTO 240
```

We arrive here if no legal key has been hit. We now go back to the start of the routine and GET another key.

```
10 REM MENU SELECTION BY CURSOR CONTROLS
20 REM THIS SHOULD BE USED AS A SUBROUTINE
40 REM START OF SUBROUTINE
50 PRINT"■":REM CLEAR SCREEN
60 ROW=1:REM INTIALISE ROW
70 PRINT"  MENU ITEM #1"
80 PRINT"  MENU ITEM #2"
```

```
90 PRINT"  MENU ITEM #3"
100 PRINT"  MENU ITEM #4"
110 PRINT"  MENU ITEM #5"
120 PRINT"  MENU ITEM #6"
130 PRINT"  MENU ITEM #7"
140 PRINT"  MENU ITEM #8"
150 PRINT"  MENU ITEM #9"
160 PRINT"  MENU ITEM #10"
170 PRINT"  MENU ITEM #11"
180 PRINT:PRINT:PRINT
190 PRINT"SELECT ITEM BY"
200 PRINT"MOVING CURSOR SOUTH"
210 PRINT"FOLLOWED BY (RETURN)"
220 REM KEY CHECKING ROUTINE
230 PRINT "■":PRINT"■";:REM HOME Q KEY AND SHIFT
240 GET A$:IF A$="" THEN 240
250 IF ASC(A$)=13 THEN ON ROW GOTO 300,310,320,330,
    340,350,360,370,380,390,400
260 IF ASC(A$)=17 THEN ROW=ROW+1
270 IF ROW=12 THEN PRINT "■■ ■":PRINT"■";:ROW=1:
    GOTO 240:REM CURSOR LEFT
280 IF ASC(A$)=17 THEN PRINT "■■ ■■■";:REM CURSOR
    LEFT CURSOR DOWN
290 GOTO 240
300 REM START ROUTINE #1
310 REM START ROUTINE #2
320 REM START ROUTINE #3
330 REM START ROUTINE #4
340 REM START ROUTINE #5
350 REM START ROUTINE #6
360 REM START ROUTINE #7
370 REM START ROUTINE #8
380 REM START ROUTINE #9
390 REM START ROUTINE #10
400 REM START ROUTINE #11
```

MENU SELECTION BY FUNCTION KEY

It is quite possible to use the function keys to select items also. Here is a routine which will allow you to select menu items by use of the 8 function keys.

How it works:

The function keys numbered 1-8 have the following ASCII values:-

Function key 1 = 133
Function key 2 = 137
Function key 3 = 134
Function key 4 = 138
Function key 5 = 135
Function key 6 = 139
Function key 7 = 136
Function key 8 = 140

The ASCII value range of the function keys is therefore from 133 to 140. We can scan the keyboard with a GET$ command to see whether or not a function key has been pressed.

Assuming that a list of menu items has been sent to the screen, here is the subroutine that is required to choose any of the 8 items on the list:-

```
150 GET A$: IF A$="" THEN 150
```

We scan the keyboard for an entry by the user. The program will wait at this line until a single keyboard character is entered at the keyboard.

```
160 IF ASC(A$)<133 OR ASC(A$)>140 THEN 150
```

We only want to know whether or not a function key has been pressed. We therefore check the ASCII value of any key to make sure that it is a function key. If the key struck is not a function key, we send the program back to line 150 and await for another key.

```
170 FUNCTNKEY=ASC(A$)
```

We arrive at line 170 only when a function key has been entered. We now give the ASCII value to the variable FUNCTNKEY.

```
190 ON FUNCTNKEY-132 GOTO R1,R5,R2,R6,R3,R7,R4,R8
```

The value in FUNCTNKEY is a number between 133-140. To be able to use this number in an ON-X-GOTO command we must subtract 132 from the variable so that the number returned will be in the range of 1-8.

The values returned *after* 132 is subtracted from the value in FUNCTN-KEY is as follows:-

Function key 1 hit: value returned = 1
Function key 2 hit: value returned = 5
Function key 3 hit: value returned = 2
Function key 4 hit: value returned = 6
Function key 5 hit: value returned = 3
Function key 6 hit: value returned = 7
Function key 7 hit: value returned = 4
Function key 8 hit: value returned = 8

We must therefore juggle the line numbers following the GOTO statement so that they will be the same position as the value of the function key struck. The variables R1-R8 refer to the function key numbers entered from the keyboard.

```
10 REM MENU SELECTION BY FUNCTION KEY
30 REM THIS MAY BE USED AS A SUBROUTINE
40 PRINT"□":PRINT:PRINT:REM CLEAR SCREEN
50 PRINT "F1 = MENU ITEM #1"
60 PRINT "F2 = MENU ITEM #2"
70 PRINT "F3 = MENU ITEM #3"
80 PRINT "F4 = MENU ITEM #4"
90 PRINT "F5 = MENU ITEM #5"
100 PRINT "F6 = MENU ITEM #6"
110 PRINT "F7 = MENU ITEM #7"
120 PRINT "F8 = MENU ITEM #8"
130 REM
140 REM KEYBOARD CHECKING ROUTINE
150 GET A$:IF A$="" THEN 150
160 IF ASC(A$)<133 OR ASC(A$)>140 THEN 150
170 FUNCTNKEY=ASC(A$)
```

```
180 REM VARIABLE FUNCTNKEY NOW HOLDS THE ASC VALUE
    OF FUNCTION KEY JUST STRUCK
190 ON FUNCTNKEY-132 GOTO 200,220,240,260,210,230,
    250,270
200 PRINT "■■■■ THIS IS FUNCTION KEY 1":GOTO 300
210 PRINT "■■■■ THIS IS FUNCTION KEY 2":GOTO 300
220 PRINT "■■■■ THIS IS FUNCTION KEY 3":GOTO 300
230 PRINT "■■■■ THIS IS FUNCTION KEY 4":GOTO 300
240 PRINT "■■■■ THIS IS FUNCTION KEY 5":GOTO 300
250 PRINT "■■■■ THIS IS FUNCTION KEY 6":GOTO 300
260 PRINT "■■■■ THIS IS FUNCTION KEY 7":GOTO 300
270 PRINT "■■■■ THIS IS FUNCTION KEY 8":GOTO 300
300 FOR DD=1 TO 1000:NEXT DD
310 GOTO 40
```

TIMING INPUT FROM THE KEYBOARD

By using the real-time clock in the Commodore 64 it is possible to control the time allowed for the user to input information. The addition of this routine to your games and simulations will add excitement and suspense.

How it works:

There are two variables which are used by the Commodore 64 for use with the real-time clock. The two variable used are TI and TI$. The variable that we will mainly be concerned with here is the variable TI. This variable is incremented 60 times a second. When using the Commodore 64, you can at any time check the number of seconds that have passed since TI was initialised to zero by typing in PRINT INT(TI/60) This command returns the number of seconds elapsed.

Here is the subroutine that will time the user's input:-

```
40 DEFFN TIMEOUT(X)=INT(TI/60)
```

We define a function TIMEOUT so that when invoked it will return a value equal to the number of seconds held in the variable TI.

```
50 ANSWER$=""
```

The string variable ANSWER$ is 'cleaned out' and is now a null string of zero characters.

```
60 LIMIT=10
```

The variable LIMIT will hold the value L where L equals the upper time limit in seconds that will be permitted for input before the computer conditionally branches to a timeout error detection routine.

```
80 FILL$=""
```

The character string variable FILL$ is 'cleaned out' and will now contain zero characters.

```
90 TI$="000000"
```

We set the real-time clock to zero. It is very important for the timing routine to commence immediately after this command.

```
100 GET FILL$
```

We invoke the GET$ command. Whenever a key is entered then FILL$ will hold the string representation of the key entered. We do not wait at this line for input but continue on so that we can test the amount of time that has elapsed.

```
110 PRINT FILL$;
```

If a key has been pressed, it will be printed on the screen. The ";" at the end of the line ensures that each character will remain on the same screen line.

```
120 IF FILL$="" THEN 140
```

This line directly relates to the line following. If no key has been struck, we jump one line.

```
130 IF ASC(FILL$)=13 THEN LINEX
```

If we arrived here and no character had been typed, the conditional would have resulted in an error. Any string that is tested for its ASCII content must return a value greater than zero. We test to see if a RETURN (ASCII value 13) has been entered. If a RETURN has been entered, it means the user has completed his or her input, so we go to the next section, starting at LINEX.

```
140 IF FN TIMEOUT(X)> LIMIT THEN GOTO LINEY
```

We now test to see if the number of seconds returned in function TIMEOUT is greater than the time limit (value in LIMIT). if this is so we go to line LINEY and continue the program.

```
150 ANSWER$=ANSWER$+FILL$
```

As the user types in a key this character is added to the string variable ANSWER$. Whenever the program leaves this input routine ANSWER$ will contain all of the characters entered up to this point.

```
160 GOTO 100
```

We arrive here if the following conditions exist:- (A) The time limit has not been reached (B) A return key has not been entered. We therefore go back and scan the keyboard again.

```
10 REM USING THE REAL TIME CLOCK TO TIME INPUT
   FROM THE KEYBOARD
20 REM MAY BE USED AS A SUBROUTINE
30 PRINT"■":REM CLEAR SCREEN
40 DEFFN TIMEOUT(X)=INT(TI/60)
50 ANSWER$=""
60 LIMIT=10
70 REM START
80 FILL$=""
90 TI$="000000"
100 GET FILL$
110 PRINT FILL$;
120 IF FILL$="" THEN 140
130 IF ASC(FILL$)=13 THEN 190
140 IF FN TIMEOUT(X)>LIMIT THEN GOTO 170
150 ANSWER$=ANSWER$+FILL$
160 GOTO 100
170 REM THE PROGRAM COMES TO THIS LINE IF TIME HAS
    RUN OUT
180 PRINT TAB(18) "■■■■■■■■■■■TIMEOUT":STOP:REM
    DOWN CURSOR
190 REM THE PROGRAM COMES HERE IF THE KEYBOARD
    INPUT IS COMPLETE BEFORE THE
200 REM TIME LIMIT IS UP
205 REM ANSWER$ HOLDS THE CHARACTER STRING THAT
    HAS BEEN TYPED IN
210 PRINT TAB(16) "■■■■■■■■■■■■" ANSWER$:REM
    CURSOR DOWN
```

FAST CURSOR POSITIONING BY PRINT STATEMENT

If you are getting tired of having to POKE the screen for fast graphics, here is an easy method using alphanumeric strings.

One of the advantages of the Commodore 64 string handling capabilities is that a string can contain cursor moving commands such as (DOWN) and (HOME). These commands can very easily be placed in a character string and PRINTed so as to rapidly move the cursor around the screen.

How it works:

The screen represents 1000 locations comprising 25 rows each 140 columns wide. To place a cursor anywhere on the screen all we need to do is have alphanumeric variable string. The first one we will call ROW$. It will contain (DOWN) cursor commands for placing the cursor at any row. The second alphanumeric string we will call COLUMN$ and it will contain (RIGHT) cursor commands for placing the cursor at any column. The string ROW$ should contain a (HOME) control followed by 24 (DOWN) controls. The string COLUMN$ should contain 39 (RIGHT) controls. By printing the leftmost n characters of these strings, we can now place the cursor anywhere on the screen.

For example to place the character "*" at row 5 and column 10 the command would be:-

```
PRINT LEFT$(ROW$,5);:PRINT LEFT$(COLUMN$,10);"*";
```

Warning

If you print a character in the lowest right hand corner the screen will scroll upwards, so it is wise to leave that location vacant.

The little demonstration program TWINKLING STARS shows how by using these commands, fairly fast graphics are quite easy to achieve.

```
10 REM CURSOR POSITIONING WITH STRINGS
30 POKE 53280,0:POKE 53281,2:REM BLACK BORDER RED
   SCREEN
60 PRINT "■":REM CLEAR SCREEN
70 ROW$="■■■■■■■■■■■■■■■■■■■■■■■■":REM HOME
   FOLLOWED BY 24 DOWN CURSOR
80 COLUMN$="■■■■■■■■■■■■■■■■■■■
```

DIET HELPER

For those of you who would like to keep the pounds/kilos off here is a little program that may help. All you need do is type in the calorie ceiling for a meal. You then enter the number of ounces that you wish to use of any particular food and the computer keeps a tally of the calories for each food used. Once the calorie ceiling is reached, the computer will print out a listing of the foods and their caloric content.

How it works.

Calories remaining are worked out at line 320 (C=C-S) where the variable C equals the updated amount of calories remaining and the variable S equals the amount of calories for a particular item.

The contents of variable S are calculated at line 280 (S=V(F)*O) where V(F) holds the number of calories for item F and variable O holds the number of ounces that the user wishes to use for that particular food. The data statements at lines 700-710 hold the names of the various foods. These names are read into an array N$(1-5). The data in line 730 holds the information relating to the calories per ounce of each of the foods. These figures are read into array V(1-5).

The program can very easily be extended to include as many items of food as you require. You must dimension the arrays at lines 90-100 to equal the maximum number of food items and calorie values in the data statements. The FOR-NEXT loops in lines 110-120 would be changed to equal the maximum number of food items, and the error check at line 220 would be changed to equal one more than the number of food items in the array.

```
10 REM *** DIET HELPER ***
50 POKE 53280,2:POKE 53281,4
60 PRINT "■";:REM WHITE
80 GOSUB 550
90 DIM T$(10),T(10),X(10)
100 DIM N$(5),V(5)
110 FOR I=1 TO 5:READ N$(I):NEXT I
120 FOR I=1 TO 5:READ V(I):NEXT I
130 PRINT:PRINT TAB(6) "ENTER CALORIE CEILING  ";
```

```
140 GET A$:IF A$="" THEN 140
150 IF A$=CHR$(13) THEN C=VAL(B$):GOTO 180
160 IF ASC(A$)>46 AND ASC(A$)<58 THEN PRINT A$;
    :B$=B$+A$:GOTO 140
170 GOTO 140
180 A=1
190 PRINT:PRINT
200 PRINT "□":GOSUB 640:REM CLEAR SCREEN
210 INPUT "FOOD #";F
220 IF F>5 THEN PRINT"□□":GOTO 210:REM ERROR TRAP
    SYMBOL IS UP CURSOR
230 T$(A)=N$(F)
240 PRINT
250 PRINT N$(F);"=";V(F):PRINT "CALORIES PER OUNCE"
260 PRINT
270 PRINT "HOW MANY OUNCES";
275 INPUT O:REM LETTER O NOT ZERO
280 S=V(F)*O:REM LETTER O NOT ZERO
290 T(A)=O:REM LETTER O NOT ZERO
300 X(A)=S
310 A=A+1
320 C=C-S
330 IF C=0 THEN GOTO 390
340 IF C<0 THEN GOTO 410
350 PRINT "□":REM CLEAR SCREEN
360 GOSUB 640
370 GOTO 210
380 PRINT
390 PRINT:PRINT TAB(8) "CEILING HAS BEEN REACHED"
400 GOTO 430
410 PRINT:PRINT TAB(8) "CEILING HAS BEEN PASSED"
420 PRINT
430 IF A<3 THEN PRINT TAB(8) "DON'T BE GREEDY!!"
    :GOTO 540
440 IF C=0 THEN A=A+1
450 FOR T=1 TO 1000:NEXT T:PRINT "□":REM CLEAR
    SCREEN
460 GOSUB 670
470 PRINT TAB(10) "YOUR DIET IS:-"
480 PRINT
```

```
490 FOR J=1 TO (A-2)
500 IF T(J)=1 THEN PRINT TAB(6) T(J);"OUNCE OF ";
    T$(J):PRINT:GOTO 530
510 PRINT TAB(6) T(J);"OUNCES OF ";T$(J)
520 PRINT
530 NEXT J
540 GOTO 740
550 PRINT "□":GOSUB 640:REM CLEAR SCREEN
560 PRINT " YOUR CHOICE IS:"
570 PRINT
580 FOR J=1 TO 5:READ N$:PRINT J,N$:NEXT J
590 RESTORE
600 PRINT
610 PRINT " PRESS ANY  KEY TO CONTINUE"
620 GET A$:IF A$="" THEN 620
630 RETURN
640 REM TITLE
645 PRINT"□":REM CLEAR SCREEN
650 PRINT "■■■■■" TAB(13) "■■DIET■ ■HELPER■■":PRINT
655 REM YELLOW REV ON REV OFF WHITE
660 PRINT TAB(9) "■■CALORIES■ ■LEFT■■";C:PRINT:
    RETURN
665 REM CYAN REV ON REV OFF WHITE
670 REM MENU TITLE
680 PRINT:PRINT:PRINT:PRINT TAB(14) "■■MENU■■":PRINT
    :RETURN
685 REM YELLOW REV ON REV OFF
690 REM MENU ITEMS DATA
700 DATA MACKEREL,BEANS,PEAS
710 DATA POTATOES,BANANAS
720 REM CALORIE VALUES
730 DATA 29,2,14,22,80
740 END
```

RUNS PER OVER

For those of you who like watching limited over cricket, here is just the program to give you an over by over analysis of the game. The program prompts for information on the number of overs played, and the score made by the opposing side. All you need do from then on is enter a number from 1-6 representing the runs scored after each ball, and the computer does the rest providing you with information on overs and balls remaining and runs needed to win.

How it works.

The formula for the run rate is:-

```
INT(((T/X)*6)*100+.5)/100
```

The variable T equals the winning score required. The variable X equals the number of balls remaining. The above formula is found at line 410 of the program.

A powerful error-input checking routine is found at line 240 where the ASCII value of the key struck is check to make sure that a number in the range 1-6 has been struck. The subroutine at line 480 emulates a flashing cursor when the GET$ command is invoked at line 230.

```
10 REM RUN RATE RECORDER
50 POKE 53280,4:POKE 53281,11
60 PRINT "□":REM CLEAR SCREEN
70 GOSUB 450
80 PRINT TAB(4) "RUNS SCORED BY FIRST SIDE";
90 INPUT T:PRINT
100 T=T+1:W=T
110 PRINT TAB(4) "OVERS FOR INNINGS";
115 INPUT O:PRINT:REM LETTER O NOT ZERO
120 PRINT TAB(4) "ENTER NAME OF SECOND SIDE":PRINT
125 PRINT TAB(4);:INPUT SIDE2$:PRINT
140 PRINT TAB(10) "■■RUNS■■ ■REQUIRED■■ =■";W;"■
145 REM CYAN REV ON REV OFF REV ON REV OFF YELLOW
    WHITE
```

```
150 PRINT
160 PRINT TAB(10) "■■AT■■";INT((T/O)*100+5)/100;"
    ■■PER■■ ■■OVER■■"
165 REM CYAN REV ON YELLOW REV OFF REV ON CYAN REV
    OFF REV ON REV OFF WHITE
170 PRINT:PRINT
180 X=O*6:REM BALLS PER INNINGS
190 FOR BALLINOVER=1 TO 6:REM WORKS THROUGH AN OVER
200 PRINT:PRINT
210 PRINT TAB(10) "■■NEXT■■ ■■BALL■■ ";
215 REM REV ON PURPLE REV OFF REV ON REV OFF
220 PRINT "■■SCORED?■■";:REM REV ON REV OFF
230 GET SC$:GOSUB 480:IF SC$="" THEN 230
240 IFASC(SC$)>47ANDASC(SC$)<55THENSC=VAL(SC$):
    PRINTSC:FORD=1TO500:NEXT:GOTO260
250 GOTO 230
260 T=T-SC
270 PRINT "■":GOSUB 450:REM CLEAR SCREEN
280 Y=INT((X/6)-.1):REM NUMBER OF COMPLETE OVERS
290 X=X-1
300 IFT<1 THEN PRINT TAB(8) SIDE2$;" HAVE WON":
    GOTO 490
310 PRINT TAB(8) "TOTAL NEEDED =";W:PRINT
320 PRINT TAB(12) "■■SCORE■■ =";W-T:PRINT:PRINT
325 REM REV ON YELLOW REV OFF
330 PRINT TAB(8) "■■RUNS■■ ■■REQUIRED■■ =■■";T;"■■"
335 REM YELLOW REV ON REV OFF CYAN WHITE
340 B=6-BALLINOVER
350 PRINT
360 PRINT TAB(8) "■■OVERS■■ ■■REMAINING■■ =■■";Y
365 REM YELLOW REV ON REV OFF REV ON REV OFF CYAN
370 PRINT
380 PRINT TAB(8) "■■BALLS■■ ■■REMAINING■■ =■■";B
385 REM YELLOW REV ON REV OFF REV ON REV OFF CYAN
390 PRINT
400 IF X=0 THEN PRINT "■":GOSUB 450:PRINT TAB(8)
    SIDE2$;" HAVE LOST":GOTO 490
405 REM CLEAR SCREEN
410 PRINT:PRINT TAB(8) "■■RUN■■ ■■RATE■■ =■■";INT
    (((T/X)*6)*100+.5)/100
```

```
415 REM REV ON YELLOW REV OFF REV ON REV OFF CYAN
420 NEXT BALLINOVER
430 O=O-1:GOTO 100:REM LETTER O NOT ZERO
440 PRINT "■":GOSUB 450:REM HOME
450 PRINT TAB(14) "■■RUN■■ ■RATE■■■":PRINT:PRINT:
    PRINT
455 REM REV ON YELLOW REV OFF REV ON REV OFF WHITE
460 RETURN
480 PRINT "■■";:FOR I=1 TO 200:NEXT:PRINT"■■ ■■";:
    FOR I=1 TO 200:NEXT:RETURN
485 REM WHITE SYMBOL FROM PLUS KEY USE COMM KEY
    LEFT CURSOR
490 END
```

CREATING YOUR OWN GAMES

You may well find that, after working through the games in this book, you are inspired to create some similar programs of your own. However, ideas for great games may not spring quickly to mind.

Therefore, Neville Predebon has devised a set of game scenarios, which you can use as starting points for creating your own games. There is no reason to stick rigidly to the outlines he presents. You should, instead, use them as starting points for your own ideas.

No. 1 - Three chances

This is a dice game, using three dice. Any number of people can play. For each round, each player picks a number between 1 and 6, not revealing it. He or she then rolls the dice. If the number comes up once, it is doubled and added to the score. If it comes up twice, the sum is trebled and added to the score. If the choice comes out on all three dice, the sum is quadrupled and added to the score. If the number doesn't appear at all, then the player is penalised three times his amount. The starting score is 100, and the game finishes when either a winner emerges after the other players have struck zero, or when a player gets 500.

No. 2 - Matchup

Four people can play this game. Each player chooses six random numbers between 1 and 100, not revealing them to the others. The computer then gives 10 random numbers between 1 and 100. For each number that a player and the computer have in common, that number is added to the score. The game ends after 10 rounds, the winner being the player with the highest score. The starting score is 0.

No. 3 - Fogspot

Three people can play this game. An alien is hiding in a thick fog, and

the players must try to capture it by the use of coordinates. If they hit it exactly, the alien is caught and the game ends. If they are three or less off either coordinate, the alien traps and kills them. The game ends when either the alien is caught or a survivor is left in the group of hunters. For every shot that player is not killed, he gains a point. The fog should be 50 by 50.

No. 4 - Time's up

Up to six people can play this game. Each player thinks of an hour of the day, and all the times are written down. Four dice are rolled. The number (sum) that comes up is taken as being an hour (13 = 1 p.m., 7 = 7 a.m., etc.) and the player with the time closest to the outcome is the winner of the round. The game ends when the dice roll either midday or midnight, and the overall winner is the player with the most round wins.

No. 5 - Nightmines

This game, played between you and the computer, has you in a large minefield on a moonless night. You must get through it to deliver a secret message to your commander. Not all the mines are fatal - some will only blow a limb off - and some are unarmed. As well as mines you have to deal with trees, potholes and a swamp. You are alerted to a mine's proximity by a buzzing noise. You have a set time to get through the field, after which the mines are automatically detonated.

No. 6 - Last unlucky

In this game, two players are locked in a game of logic with the computer. The computer picks a random number between 50 and 150. The two players take turns to remove anything up to three numbers from the result. The loser is the player forced to remove the last number, and the other player (the winner) is given a score equal to the amount thought of by the computer. The game ends when the first loser passes the highest score obtained by the other player.

No. 7 - Three mile island

A nuclear power station has gone critical and you must calm it by using

stabilising rods. If you can bring the radiation count below 500, the station is safe. If the count passes 2000, it will blow. For the amount of time it is critical, the station will lead radiation at an ever-increasing rate. If not used correctly, the stabilising rods will decompose. If you are left with less than four rods out of a starting total of ten, the station will blow.

No. 8 - Hellish sunset

A mid-Pacific volcano has erupted, destroying most of the island it is on. It threatens to detonate the Earth's core unless the lava flow to the chimney is cut off. New lava tunnels are formed almost as fast as you block them. The volcano will not blow at all if you reduce the tunnels to five or less, and the Earth is doomed if the tunnels number more than twenty. In between these extremes the core is gradually getting hotter and the after-eruptions more violent.

No. 9 - Island one

In this game, a war has erupted between Earth and the first orbiting space colony due to Earth cutting off vital supplies. You have a twofold task - to protect the Island or defeat Earth while a mission is sent to mine the metals needed from the asteroid belt. You defeat Earth if you manage to knock out her assault base, which is continually moving in atmosphere. Earth is blasting the station's shields, one by one, and is getting dangerously close to the central power reactor - which will devastate the station if it is hit. If the mining fleet returns with the supplies before the reactor goes, a ceasefire will be declared.

No. 10 - Serpentine swamp

You are in the middle of a large African swamp, in a boat with a faulty engine that only works in fits. There are large and deadly snakes and bull crocodiles, which you must fend off in between trying to cross the swamp in spurts. The engine only works spasmodically, and will break down altogether if used for too long at a time. The beasts will surround your boat each time you stop, but if they are irritated or hungry enough they will attack each other, giving you a chance to escape (providing the engine works). If you reach safe land you will have no further troubles.

No. 11 - Trans-Atlantic terror

A terrorist group has planted a number of bombs on board a trans-Atlantic jetliner. You, as the captain, must find them before landfall. The bombs are set to detonate in five hours, while New York is six hours away. The bombs can be disarmed if they can be found - there are four on board, with several other false-alarm packages.

No. 12 - Out of the silent sky

A nuclear-powered satellite has slipped its orbit and is dropping toward Earth at an increasing speed. You are the pilot of a strike craft sent to destroy it as it enters atmosphere. If it crash-lands it will release its power pack and contaminate the surrounding area. You have three missiles to use against it - if they don't hit your only recourse is to ram it with your fighter to destroy it. The higher up it is destroyed, the less danger there is to the area below, and the smaller the area of danger.

No. 13 - Surround

You are a figure in the centre of the screen. The computer attempts to trap you by drawing a maze around you (you can move to escape the walls). The game ends when either you manage to reach the edge of the screen without being hit by the lines, in which case you win, or when the walls enclose and/or touch you. For each time you reach the border – where the lines have not been – you are allotted five points.

No. 14 - Intercept

You are a point set in a randomly-chosen area of the screen, and you must avoid two other points (also randomly situated) as they try to intercept and strike you. There are no easy ways to escape in this game; your score depends on how long you can remain on the run. No matter where you move, the interceptors change direction to meet you. The interceptors travel more rapidly the longer the game progresses, while you gain teleport abilities after a score of 250.

No. 15 - Plague force

You are a point randomly situated near the centre of the screen. The rest of the screen rapidly begins to fill with plague bacteria, and you must escape them. Your only hope is to reach the border, where you can receive immunisation. Your score depends on how long you can escape the plague, and is doubled when you reach the medical station. If you are struck by a bacterium while in flight you have only a short while to live (10 points) unless you reach the medical station beforehand. The score is increasing with each moment, and you are granted bursts of temporary immunity after a score of 200.

No. 16 - Prisoner

A group of convicts is attempting escape by climbing a ladder and going over the wall. You are the only warden on duty at the time, and must try to stop them escaping until other guards arrive. For every rung each prisoner climbs, you are penalised five points and this amount (in dollars) will be deducted from your pay. There are seven convicts, and you will lose your job if all escape. You attempt to foil their bid by sawing off the legs - each time you catch an escapee you will be rewarded 1000 points ($1000).

No. 17 - The final countdown

Any number of people can play this game. Three dice are used. Each player determines his score in this way - he rolls all three dice, then sets aside the highest one. He rolls the remaining two, and again sets aside the highest one. The number on the die remaining is taken as being the score, and is added to that player's total score. The winner is the player with the highest score after ten rounds.

No. 18 - Odds and evens

This game is for three people. Using two dice, each player takes a turn in casting a roll. If the sum of the outcome is even, the amount is added to the running total. If it is odd, the higher of the two numbers is deducted from the running total or the starting total of ten. The game ends when one player reaches 100 and becomes the winner.

No. 19 - Don't pay the ferryman

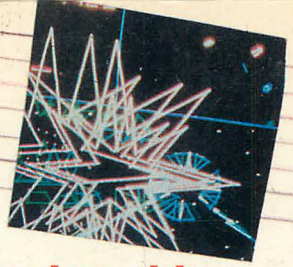
You are on a barge, crossing the River of Hopelessness on your way to Hell, with Death as your ferryman. You have to try to bribe him to turn around and make for the Shore of Salvation using jewels floating on the river. Not all jewels are within reach - you will have to retrieve some by swimming, but if you cannot make it back to the boat you are doomed. The ferryman will turn back if you manage to offer him over 5000 carats worth of diamonds and sapphires, but your chances will decrease if you offer him rubies or oyster pearls.

No. 20 - The stars, like dust

You are a lone occupant of a starship capable of intergalactic travel through teleportation. You have to cross several galaxies to deliver a piece of starshine to the king of a distant world. It takes a minimum of five jumps to cross one galaxy. There is a danger that you could accidently jump into a star or a black hole or the inside of a planet, and these dangers increase as you near the centre of the universe. You must use the powers of the star-fragment to guide your way through space - it will brighten if danger is imminent. You will be rewarded the king's daughter when you arrive, but this will not apply if you are overdue by more than seven moons. The fragment will explode if it gets too bright.

No. 21 - Emergency

In this game, you are the driver of an ambulance carrying a seriously wounded accident victim to hospital. Using your siren you must fight your way through heavy traffic and busy intersections. Your patient is not strapped in properly and you will have to stop a few times to pick him up when he slips out. If you do not reach the hospital within a certain time, the patient will die.



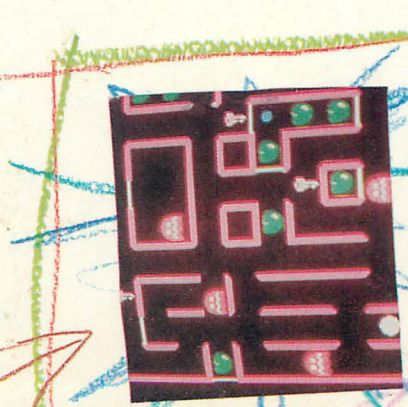
No matter what kind of games you enjoy playing, this generous collection of programs for the Commodore 64 puts you right on target for excitement.

Robert Young and Paul Copeland have brought together a set of games which will make you glad you invested in a Commodore 64.

Take part in adventures in imaginary environments, get your reflexes working in arcade-like games, pit your brain against the computer's electronic one in hard-to-win board games, and have a lot of fun learning how to improve your own computer programming at the same time.

A good computer program starts with one original idea, which is nurtured and allowed to grow in its own good time. From the moment the first keys were pressed on the Commodore 64, to the time the final program started to roll off the printer, Robert Young and Paul Copeland were undergoing a process of discovery, with the Commodore 64 bringing its own influence to bear, shaping the great programs in this book into their final form.

Come on in, the game-playing is fine!



U.K. £2.95
AUST. $12.95
(recommended)

U.K.
AUST.
[reco

Games
ISBN 0 14
00 7890 8